



תיק פרויקט – Snake DQN

שם התלמיד: יואב גלעד

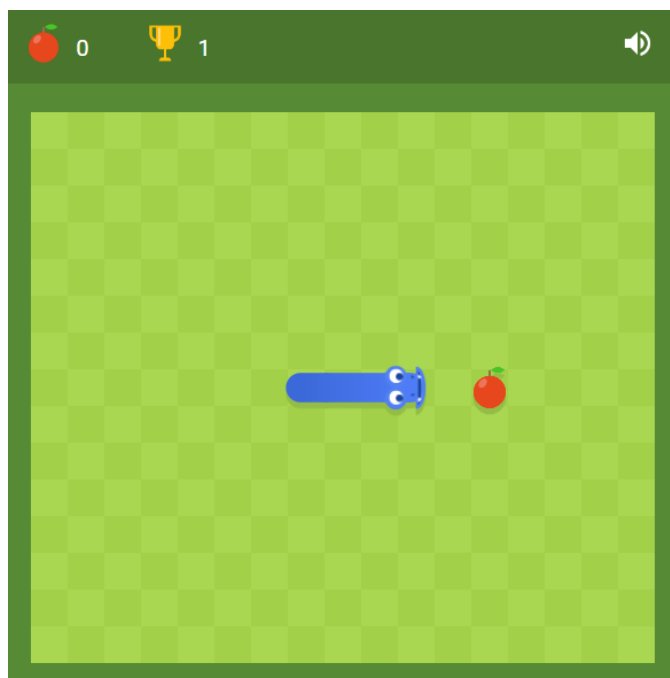
ת.ז. התלמיד: 216875922

בית ספר: תיכון המושבה

שם המנחה: ענבר מור

שם החלופה: למידת מכונה

תאריך הגשה: ?



תוכן עניינים

1.....	Snake DQN – תיק פרויקט
2.....	תוכן עניינים
3.....	מבוא
4.....	דגשים חשובים לקריאת תיק הפרויקט
4.....	רקע תיאורטי
6.....	מבנה הפרויקט
7.....	ייצור הנתונים
8.....	שדרוגים לסביבה Snake
10.....	תיג הנתונים
11.....	בניית ואימון המודל
12.....	בניית המודל
14.....	היפר-פרמטרים וכוונן
16.....	תוצאות אימון המודל בדוגמה
19.....	ייעול אימונים
24.....	יישום
25.....	מדריך למפתח
26.....	environments.py
34.....	agents.py
40.....	evaluation.py
44.....	logs.py
45.....	graphs.py
47.....	emails.py
48.....	mytime.py
49.....	training.py
55.....	testing.py
57.....	view game.py
58.....	מדריך למשתמש
59.....	רפלקציה
60.....	ביבליוגרפיה
61.....	נספחים

מבוא

Reinforcement Learning (RL) הוא ענף בלמידת מכונה המתמקד בייעול התנהגות וקבלת החלטות בסביבה מסוימת, בפרויקט שלי במשחק Snake. היעד הוא להגיע למדיניות (policy) המתארת התנהגות אופטימלית בסביבה. הפרויקט כולל תשתית לאימון סוכן RL באלגוריתם DQN, ואת המשחק Snake כסביבה לדוגמה לשימוש. ניתן לבנות ולהוסיף סביבות נוספות. מעבר לכך, הפרויקט כולל גם אפשרויות שליטה על פרמטרים רבים ופיצ'רים נוספים להאצת התכנסות ושיפור ביצועים, שמירת והצגת נתוני אימון ויישום, ואנימציה לבחינה ויזואלית של יישום המודל בסביבה.

Snake הוא משחק מחשב לשחקן יחיד, בו השחקן שולט בנחש שנע במשטח דו ממדי בצורת "רשת" של ריבועים (grid). במשטח מפוזרים תפוחים ובכל פעם שהנחש מגיע לתפוח הוא אוכל אותו, גופו מתארך ונוצר תפוח חדש במקום אחר. מטרת השחקן היא לאכול כמה שיותר תפוחים מבלי להיתקע בעצמו או בקירות התוחמים את הלוח, שכן זוהי פסילה. התארכות הנחש מובילה לרמת קושי עולה ככל שהמשחק מתקדם, במיוחד אם גודל הלוח קטן.

בחרתי במשחק Snake בשל פשטותו, משום שהדבר מאפשר לי לבדוק את היעילות של שדרוגים רבים ומגוונים לגרסה הבסיסית של DQN ואת ההשפעה של פרמטרים שונים על ההתכנסות, תוך שמירה על עלות חישובית מינימלית ועל evaluation פשוט. הסביבה Snake מהווה צעד ראשוני בתהליך ההתקדמות לעבר שימוש ב-RL בכלל, וב-DQN בפרט, למקרים יותר ויותר פרקטיים ורלוונטיים לבעיות בעולם האמיתי.

קיימים מימושים רבים של DQN ופרויקטים רבים של המשחק Snake באלגוריתם זה, אך לא מצאתי אף פרויקט שמאפשר שליטה בפרמטרים ופיצ'רים בכמות שמגיעה למימוש שלי. מימנתי את האלגוריתם תוך לקיחת השראה מהמקורות קיימים, אך אני יצרתי את עיקר הקוד ואת הפיצ'רים הנוספים. התבססתי על מאמר [1] לערכים אופטימליים לחלק מהפרמטרים, כגון מקדם הנחה (gamma) וערכי התגמולים (rewards). ערכים אלו נתנו לי בסיס להתחיל את האימונים ממנו, ולהתמקד בפרמטרים ושדרוגים אחרים שהעדפתי להתעסק בהם.

קהל היעד של הפרויקט הוא מפתחים. הפרויקט מספק מימוש לדוגמה, או לפחות מימוש חלקי של DQN עם יותר פיצ'רים ואפשרויות ממרבית המימושים הקיימים. מפתחים יכולים להשתמש בסביבה הקיימת בשביל חקר ביצועים על קונפיגורציות של פרמטרים ופיצ'רים של DQN לפני שמנסים לשלב אותם בפרויקטים אחרים, או להשתמש בחלקים הדרושים מהמימוש.

חשוב לציין שמאחר ש-Snake הוא משחק פשוט, ניתן להגיע לביצועים מעולים באמצעות אלגוריתמיקה קלאסית, ללא למידת מכונה. אך יש לזכור שמטרת הבחירה במשחק Snake היא שישמש כסביבה לביצוע RL, לכן הפרויקט מתמקד בפיתוחו באמצעות RL, וספציפית DQN.

דגשים חשובים לקריאת תיק הפרויקט

מלבד תוכן העניינים, לאורך הקובץ קיימות התייחסויות למקומות אחרים בו. לנוחותכם, הוספתי קישורים המסומנים ב-*italic* ולחיצה עליהם תוביל למקום המוזכר.

ידע קודם אודות פעולתן של רשתות נוירונים הכרחי להבנת קובץ זה, ולא יפורט בתיק. מומלץ ידע ב-Reinforcement Learning (RL) וב-DQN, אך הוספתי רקע תיאורטי בסיסי על כך וכן פירוט מעמיק יותר על שדרוגים במקומות הרלוונטיים בהמשך. מוזמנים גם להיכנס לקישורים **בנספחים** להסברים מפורטים יותר לחיזוק ההבנה.

רקע תיאורטי

RL – הגדרה כללית

RL הוא ענף בלמידת מכונה המתמקד ביעול התנהגות וקבלת החלטות בסביבה מסוימת, כגון משחקי מחשב ורובוטיקה. בפרויקט שלי במשחק Snake. היעד הוא להגיע למדיניות (policy) המתארת התנהגות אופטימלית בסביבה. הלמידה נעשית על ידי ניסוי וטעיה של "סוכן" שפועל ב-"סביבה". באלגוריתם DQN הסוכן הוא רשת נוירונים עמוקה שנקראת Q-Network.

סביבה

לעיתים קרובות משתמשים בסימולציה ממוחשבת כסביבה, במיוחד עבור משחקי מחשב. סביבה שמאפשרת תהליך Reinforcement Learning באלגוריתם DQN צריכה לענות על מספר דרישות:

- מרחב פעולה סטטי – בכל מצב בסביבה יש אותן פעולות אפשריות (כל נוירון בשכבת הפלט של רשת ה-Q מייצג פעולה).
 - ב-Snake הפעולות הן ימינה, שמאלה, למעלה, למטה.
 - שחמט, לדוגמה, היא סביבה לא מתאימה, משום שכמות הפעולות משתנה בין מצבים, וכך גם המשמעות שלהן.
- ייצוג מצב קבוע – מבנה הקלט לרשת צריך להישאר קבוע.
- השפעת הפעולה הנוכחית תלויה רק במצב הנוכחי, לא באירועים קודמים.
- מערכת תגמולים מתאימה – תלוי במפתח.

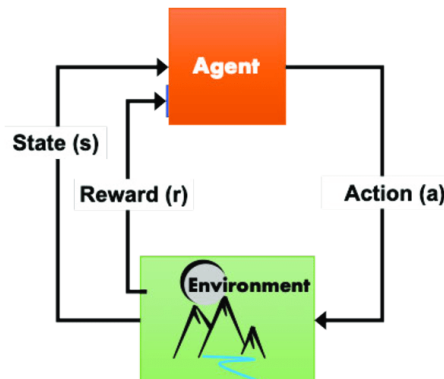
סוכן

בכל איטרציה ב-RL, הסוכן מקבל את מצב הסביבה (state) ובוחר איזו פעולה (action) לבצע. ב-DQN מדובר ברשת נוירונים עמוקה, שהקלט שלה הוא מצב הסביבה, והפלט הוא ציוני איכות (ערכי Q – פירוט בהמשך) לכל הפעולות האפשריות במצב. הפעולה עם הציון הכי גבוה היא זו שהסוכן "בוחר".

תהליך אימון הסוכן נעשה באמצעות שני תהליכים נפרדים הפועלים לסירוגין או במקביל:

1. איסוף מידע – צבירת "חוויית"

בכל איטרציה ייצור מידע: הסוכן מקבל כקלט את המצב הנוכחי בסביבה (State), ובוחר פעולה (Action). לאחר ביצוע הפעולה הסוכן מקבל פידבק מהסביבה בצורה של "תגמול" (Reward), שנקבע על ידי המפתח עבור אירועים מסוימים. תגמול יכול להיות חיובי, שלילי או ניטרלי; גודל התגמול משפיע גם הוא על האימון. הפידבק כולל גם את המצב החדש בסביבה, שימש בהמשך לתיוג החוויה שנצברה.



כל חוויה מאוחסנת בזיכרון שנקרא experience buffer, ובהמשך תשמש לאימון.

2. תיוג ואימון

בכל איטרציה אימון שולפים חוויות מהזיכרון ומתייגים אותן. ערך ה-Q הוא התגית, ומחושב באמצעות Bellman equation:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

s – state, s' – next state

a – action, a' – next action

r – reward

gamma – (בד"כ 0.9-0.99) מקדם לחשיבות הפרסים העתידיים

Q(s', a') – מוערך באמצעות חיזוי הרשת

נשים לב שהגדרת $Q(s, a)$ היא רקורסיבית, משום שהיא כוללת את $Q(s', a')$, שהוא ערך ה-Q של הפעולה הבאה. תיאורטית, הרקורסיה נעצרת כשאין עוד פעולה, כלומר כשנגמר המשחק. בפועל, תיוג המידע משתמש בחיזוי של הרשת הנוכחית עבור $Q(s', a')$, ולכן התיוג לא מדויק לחלוטין. אך רכיב ה-reward מדויק, משום שהתקבל כפידבק מהסביבה, ולכן יש ערך לתיוג זה.

בכל איטרציה מתבצע תיוג מחדש יותר מדויק מהקודם, משום שחיזוי הרשת משתפר.

מבנה הפרויקט

נהוג לחלק את שלבי למידת המכונה לשלושה חלקים:

1. איסוף, הכנה וניתוח הנתונים – Collect, Prepare, and Analyze data
2. בניית ואימון המודל – Build and train deep learning model
3. יישום – Deployment

ב-Reinforcement Learning החלוקה פחות מוגדרת, משום שייצור הנתונים ותיוגם מתבצע במקביל לאימון המודל. לכן אסביר באמצעות חלוקה זו, אך לא מדובר בהכרח ברצף כרונולוגי. להלן תמצית רצף האימון במימוש שלי, שנועדה להעניק מבט-על לפני שצוללים להסבר המעמיק על כל חלקי הפרויקט.

- תחילת הריצה: טעינת פרמטרים, בניית מחלקת הסוכן, בניית הסביבה.
- ייצור נתונים ראשוני לפני האימון, עד לכמות מינימלית.
- תחילת האימון בסבבים. כל סבב כולל:
 - איטרציות אימון:
 - קבלת המצב מהסביבה
 - חיזוי ובחירת הפעולה
 - ביצוע הפעולה בסביבה
 - קבלת פידבק
 - אחסון הפידבק בזיכרון
 - ייצור נתונים
 - שליפת "חוויית" אקראיות מהזיכרון
 - תיוג החוויות
 - תיוג נתונים
 - אימון הרשת על החוויות המתויות
 - עדכון פרמטרים דינמיים (פירוט בהמשך)
 - איטרציות מבחן:
 - קבלת המצב מהסביבה
 - חיזוי ובחירת הפעולה
 - ביצוע הפעולה בסביבה
 - שמירת ביניים של המודל ומדדי האימון והמבחן.
 - עדכון המשתמש בסטטוס האימון.
- שמירה סופית של המודל, מדדי האימון והמבחן, וייצור גרפים עבורם.
- עדכון המשתמש בסיום האימון.

ייצור הנתונים

התחלתי מלבנות את המשחק Snake כמחלקה שתפקד כסביבת אימון ויישום ממוחשבת, בה יתרחש ייצור הנתונים. שמתי דגש על מבנה מחלקה המתאים לתהליך אימון ב-DQN (ראה רקע תיאורטי).

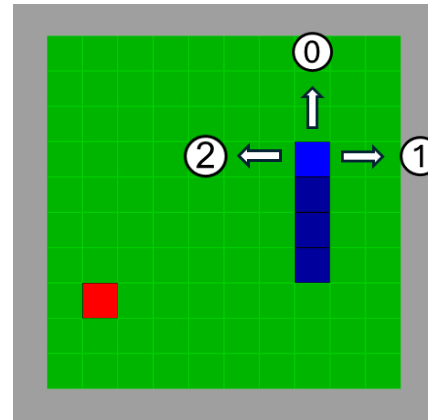
בחרתי בתנועה יחסית, כלומר שלוש פעולות המסמלות תנועה ישר, פנייה ימינה ופנייה שמאלה ביחס לכיוון הנוכחי.

בחרתי לייצג את לוח המשחק כמטריצה עם 5 ערכים שונים שמייצגים תפוח, ראש הנחש, גוף הנחש, קיר ומשבצת ריקה. הערכים מנורמלים לטווח -1 עד 1.

השתמשתי בספרייה graphics.py, שהיא ספריית גרפיקה בסיסית ויחסית איטית, אך מספיק טובה לנקודת זמן זו.

הדגמת מצב המשחק והפעולות האפשריות:

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	0	0	0	0	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	0.5	0	0	0	-1
-1	0	0	0	0	0	0	0	-0.5	0	0	0	-1
-1	0	0	0	0	0	0	0	-0.5	0	0	0	-1
-1	0	0	0	0	0	0	0	-0.5	0	0	0	-1
-1	0	1	0	0	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	0	0	0	0	-1
-1	0	0	0	0	0	0	0	0	0	0	0	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1



תגמול	מאורע
+15	לאכול תפוח
-10	להיתקע בקיר
-10	להיתקע בגוף
+1	להתקרב לאוכל
-1	להתרחק מאוכל

מימשתי את מערכת התגמולים המופיעה במאמר [1], משום שמערכת תגמולים טובה תאפשר לי להתמקד בבדיקת יעילות השדרוגים של DQN, במקום להתעסק בבניית ובדיקת מערכת מאפס. מערכת התגמולים מתוארת בטבלה:

התגמולים משמשים לתיוג, עליו אסביר בהמשך.

שדרוגים לסביבה Snake

לאחר שכבר אימנתי ובדקתי מספר מודלים, נתקלתי במספר צרכים שבעקבותיהם שיניתי את הסביבה.

סביבה וקטורית

עקב זמן אימון ממושך, יצרתי סביבה וקטורית כמחלקה VecSnake, שמאפשרת מספר משחקים במקביל על מנת להאיץ את ייצור הנתונים ואת מבחן המודל. לדוגמה, במקום לבצע 10 פעולות עוקבות בסביבה אחת, בכל איטרציית איסוף נתונים, אפשר לבצע פעולה אחת בכל 1 מתוך 10 סביבות מקבילות. ב-TensorFlow, עשרה חיזויים נפרדים איטיים הרבה יותר מחיזוי אחד על עשרה קלטים בבת אחת, לכן סביבה וקטורית הובילה לשיפור דרסטי בזמן האימון והמבחן.

קטיעת לולאות

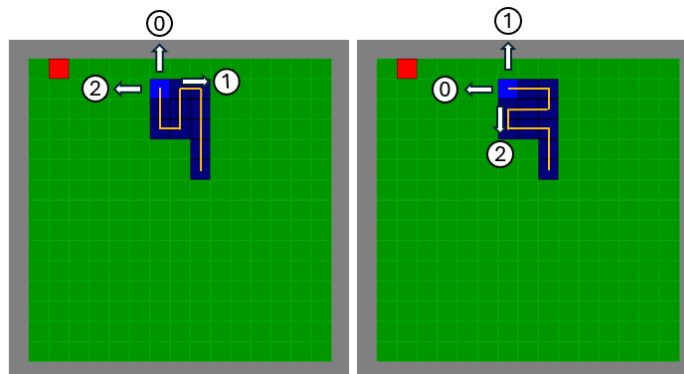
במודלים שעדיין לא התכנסו לביצועים טובים, לעיתים מתרחשת בעיה שהם חוזרים על רצף פעולות זהה המסתיים בנקודת ההתחלה, ונעים במעין "לולאה" אינסופית. במהלך האימון הלולאה נקטעת בסופו של דבר ע"י exploration כאשר epsilon גדול מ-0, אך בשלב המבחן לא מתבצע exploration. התמודדתי עם הבעיה על ידי זיהוי לולאות במשחק וקטיעה מוקדמת שלהן. אם מספר הפעולות הרצופות בהן לא נאכל תפוח או נגמר המשחק עובר סף מסוים – הסביבה תסיים את המשחק גם בלי פסילה. חשוב להשתמש בקטיעת לולאות רק בשלב המבחן ולא בשלב ייצור הנתונים, על מנת להימנע ממניפולציות שישבשו את הנתונים.

Pygame

זוהי ספריית אנימציה יותר נוחה שמאפשרת גם אינטראקציה מובנית עם משתמש אנושי, לכן עברתי להשתמש בה לבדיקת המודל ולמשחק אנושי בסביבה.

תנועה מוחלטת

שימוש בארבע פעולות אפשריות המסמלות כיוונים מוחלטים בלוח (למעלה, למטה, ימינה, שמאלה) במקום פנייה ביחס לנחש שהייתה לפני. תנועה מוחלטת שיפרה ביצועים ע"י התמודדות עם אי-ודאות במצבים זהים לכאורה, בהם מטריצת הלוח זהה אך השפעת הפעולות שונה. דוגמה למצבים זהים לכאורה היוצרים חוסר בהירות כאשר משתמשים בתנועה יחסית:



תוצאות מעבר לתנועה מוחלטת:



ניתן לראות התכנסות על רמת ביצועים גבוהה יותר עבור תנועה מוחלטת (4 פעולות) לעומת תנועה יחסית (3 פעולות).

ייתכן כי העיכוב הקטן בשיפור בתחילת האימון נגרם משום שאתחלתי חלק מהמשקולות באמצעות מודל מאומן על תנועה יחסית.

תיוג הנתונים

במהלך שלב הייצור נצברות חוויות, שנשמרות בצורה:

(state, action, reward, new state, done)

done הוא משתנה בוליאני על האם נגמר המשחק.

על החוויות מהזיכרון מבוצע experience replay: בכל איטרציית אימון שולפים חוויות מה-
experience buffer, מתייגים אותן, ומאמנים עליהן את המודל.

לכל חוויה מחושב ערך ה-Q ע"י Bellman equation (הסבר ברקע התיאורטי):

$$Q(s, a) = \begin{cases} r & \text{if done} \\ r + \gamma \max_{a'} Q(s', a') & \text{if not done} \end{cases}$$

בכל חוויה נתון state אחד עם action אחת בלבד וה-reward עליו, לכן ניתן לחשב רק את ערך ה-Q מפעולה זו. אך בהינתן state, הרשת חוזה את ערכי ה-Q של כל ה-actions האפשריים במצב (במקרה הזה 4).

מאחר שנתון פידבק רק על פעולה אחת, התיוג מתבצע באופן חלקי, כך שרק הערך של הפעולה שקרתה מתויג.

נסמן q_a = ערך מחיזוי, Q_a = ערך מחישוב. אם נבחרה הפעולה action=2, נקבל:

הקלט הוא: $x = state$

חיזוי הרשת הוא: $\hat{y} = [q_0, q_1, q_2, q_3]$

התגית היא: $y = [q_0, q_1, Q_2, q_3]$

כך מתייגים כל חוויה ב-minibatch של מספר חוויות.

* גם ערך ה-Q שמחושב כולל רכיב של חיזוי, שהוא $Q(s', a')$. חיזוי זה לא מתבצע על ידי הרשת העיקרית של הסוכן, אלא על ידי target network. רשת זו היא גרסה מוקדמת יותר של המודל, והשימוש בה מייצב את האימון משום שכך חישוב התגית לרשת העיקרית לא תלוי בפלט של עצמה. רשת המטרה מסתנכרנת עם הרשת העיקרית כל כמות מסוימת של איטרציות אימון (התדירות נקבעת לפי פרמטר), כלומר מקבלת את המשקולות שלה. בסנכרון ניתן לבצע עדכון רך (soft update) שמרכיב את העדכון מהמשקולות של שתי הרשתות, לפי האחוזים הנקבעים בפרמטר tau. עדכון קשיח הוא העתקה מלאה של המשקולות העיקריות, ללא שימוש במשקולות של רשת המטרה.

בניית ואימון המודל

מימוש מחלקת הסוכן כלל הרבה אתגרים. היו לי בעיות בהתקנה ובקונפיגורציה של TensorFlow, אך הצלחתי לאחר מספר ימים. תהליך מימוש המחלקה הוביל אותי למסקנה שההבנה שלי לגבי האלגוריתם לא הייתה מספקת בשביל ליישם אותו לחלוטין, לכן חזרתי ללמוד תוך מחשבה על איך איישם בקוד כל חלק באלגוריתם.

בחירת מבנה הרשת והיפר-פרמטרים כללו שילוב של ערכים נפוצים ממקורות מהם למדתי, והרבה ניסוי וטעיה שלי לכוון שלהם תוך שימוש בכלי מדידה והערכה שיפורטו בהמשך.

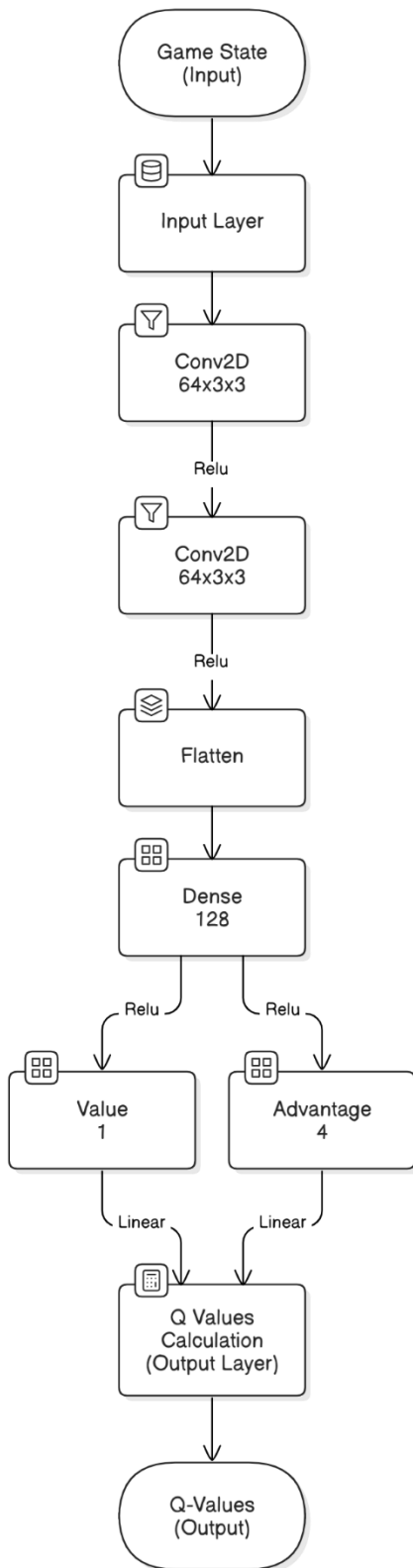
התחלתי מאימון רשתות MLP, אך לא ניכר שום שיפור ברמת המודל. לאחר debugging ממושך שמתי לב לתופעה מוזרה בנתונים: היו כפילויות רבות של מצבים ב-*experience buffer*. הבעיה נבעה משמירת פוינטר למטריצת המשחק במקום העתקה שלה. לאחר תיקון ניכר שיפור ברמת המודל, אך עדיין היה פער משמעותי מרמה אנושית סבירה.

התחלתי תהליך ארוך של חקר ביצועים שבאמצעותו ביצעתי hyper-parameter tuning, שיניתי את ארכיטקטורת הרשת, והוספתי שדרוגים לסוכן, לסביבה ולאגוריתם שמטרתם לשפר את הביצועים ולהאיץ את האימון. להלן תמצית השינויים שהכנסתי לאורך הזמן (סדר לא כרונולוגי):

מטרת השינוי			שם השינוי	היכן השינוי בא לידי ביטוי
מדידה, בקרה והפקת תובנות ליישום	האצת התכנסות	שיפור ביצועים		
	✓	✓	Experience replay	סוכן
		✓	CNN	
		✓	Dueling network	
	✓	✓	Target network	
	✓	✓	Soft updates	
	✓		הגדלת מודל מאומן	
	✓		סביבה וקטורית	סביבה
✓			קטיעת לולאות	
		✓	תנועה מוחלטת	
✓			ספריית גרפיקה	
	✓	✓	כוונון פרמטרים	פרמטרים
✓			Profiling	לולאת האימון
✓			שמירת מדדי אימון	
✓			אימון מקדים למודלים מוגדלים	
	✓		Multi-Processing	
	✓		Checkpoints	
	✓		אימיילים	
	✓		Verbose	

בניית המודל

Q-Network Diagram



בסופו של דבר הגעתי לארכיטקטורת הרשת המוצגת, עליה אפרט בעמוד הבא.

```
def create_model(self, new_shape: tuple[int, int], old_model=None, action_space: int = 4):
    """ ... """

    # new model definition. should be identical to old model except input layer.
    inputs = Input(shape=(new_shape[0], new_shape[1], 1))
    x = Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(inputs)
    x = Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(x)
    x = Flatten()(x)
    x = Dense(units=128, activation='relu')(x)
    value = Dense(units=1, activation='linear')(x)
    advantage = Dense(action_space, activation='linear')(x)
    mean_advantage = tf.keras.ops.mean(advantage, axis=1, keepdims=True)
    q_values = value + (advantage - mean_advantage)

    new_model = tf.keras.Model(inputs=inputs, outputs=q_values)

    # transfer weights if given
    if old_model is not None:
        for old_layer, new_layer in zip(old_model.layers, new_model.layers):
            if isinstance(new_layer, Dense):
                # zero padding to the end of the array.
                old_weights, old_biases = old_layer.get_weights()
                pad_size = new_layer.get_weights()[0].shape[0] - old_weights.shape[0]
                padded_weights = np.pad(old_weights, ((0, pad_size), (0, 0)),
                                         mode='constant', constant_values=0)
                new_layer.set_weights([padded_weights, old_biases])
            elif isinstance(new_layer, Conv2D):
                # no padding
                new_layer.set_weights(old_layer.get_weights())

    new_model.compile(loss='mean_squared_error', optimizer='adam')
    return new_model
```

- גודל שכבת הקלט יכול להשתנות בהתאם לגודל הלוח עבורו יוצרים את המודל.
 - תחילה, הקלט עובר 2 שכבות קונבולוציה עוקבות, כל אחת עם 64 קרנלים בגודל 3x3 הנעים בפסיעות של 1, עם אקטיביציית ReLU. שכבות הקונבולוציה נועדו לזיהוי דפוסים מרחביים במשחק, וכמות המשקולות בהן לא תלויה בגודל הקלט, לכן אני מבצע העתקה מלאה שלהן במעבר בין מודלים של גדלי לוח שונים.
 - לאחר שיטוח, הקלט מועבר לשכבה לינארית עם 128 ניוונים ואקטיביציית ReLU. כמות המשקולות בשכבות הלינאריות תלויה בגודל הקלט, לכן אני מבצע העתקה חלקית שלהן במעבר בין מודלים, ומשלים באמצעות ריפוד באפסים (ניתן להשתמש באינטרפולציה, אך ברוב המקרים תיווצר הטייה לפעולה אחת ויעילות ייצור הנתונים תפחת).
 - הקלט מועבר לשני ערוצים נפרדים: Value ו-Advantage. זוהי ארכיטקטורת Dueling Network (ראה מאמר [5]), שמספרת את יעילות האימון כשלא כל הפעולות נחקרו בכל מצב, ומייצבת את הלמידה ע"י הפחתת רגישות לפידבקים "רועשים".
 - ערוץ ה-Value מעריך את התגמול הצפוי ממצב ללא תלות בפעולות, באמצעות ניוון 1 ואקטיביציית לינארית.
 - ערוץ ה-Advantage חוזה את התגמול מכל פעולה במצב, באמצעות 4 ניוונים ואקטיביציית לינארית.
 - מחסרים מערכי ה-Advantage את הממוצע שלהם על מנת לשמור רק על היחסים בין הפעולות, וסוכמים עם חזיו ערוץ ה-Value לקבלת ערכי Q.
 - תוצאת צירוף הערוצים היא הפלט הסופי של הרשת.
- מאחר שחזיו ערכי Q היא מטלת רגרסיה, אני משתמש בפונקציית השגיאה Mean Squared Error:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

n שווה לכמות החוויות שנשלפות מהזיכרון בכל איטרציית אימון.

להגדרת y ו- \hat{y} ראה תיוג הנתונים.

מאחר שאני רוצה להתמקד בפרמטרים ייחודיים ל-RL, עבור אופטימיזציה להתכנסות אני משתמש ב-Adam עם ערכי ה-default של keras לפרמטרים: learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07

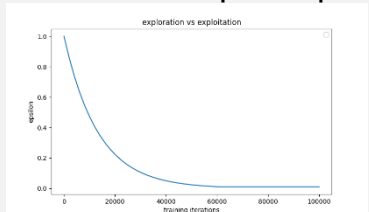
היפר-פרמטרים וכוונון

```
'15x15 4': {'OLD_ACTIONS': False,
            'HEIGHT': 15,
            'WIDTH': 15,
            'MIN_FOODS': 1,
            'MAX_FOODS': 1,
            'MIN_START_LENGTH': 4,
            'MAX_START_LENGTH': 4,

            'ROUNDS': 100,
            'FITS': 1000,
            'ENVS': 100,
            'FIT_FREQ': 1,
            'EPOCHS_PER_FIT': 1,
            'MINIBATCH_SIZE': 1000,
            'MEMORY_SIZE': 1_000_000,
            'MIN_MEMORY_SIZE': 100_000,
            'TARGET_SYNC_FREQ': 100,
            'TAU': 1,
            'GAMMA': 0.95,
            'EPSILON': 1,
            'EPSILON_DECAY': 0.999925,
            'MIN_EPSILON': 0.01,
            'DECAY_BEFORE_TRAINING': True},
```

אימנתי מודל עם ארכיטקטורת הרשת הנ"ל ועם הפרמטרים המוצגים בתמונה. מצורפת טבלה המסבירה על המשמעות של הפרמטרים השונים, ולאחריה תוצאות האימון בקונפיגורציה זו.

פרמטרים	הסבר	הערות
OLD ACTIONS	קובע אם להשתמש במנגנון התנועה הישן (תנועה יחסית) או בחדש (תנועה מוחלטת).	
HEIGHT, WIDTH	גודל הלוח הוא $HEIGHT \times WIDTH$.	
MIN FOODS, MAX FOODS	קצוות הטווח לכמות התפוחים שתהיה בסביבה בכל רגע נתון. הכמות נקבעת כמספר אקראי בטווח.	הוספתי את האפשרות הזו לסביבה אך בפועל התמקדתי באימונים עם תפוח אחד בלבד.
MIN START LENGTH, MAX START LENGTH	קצוות הטווח לאורך הנחש בתחילת המשחק, האורך הוא מספר אקראי בטווח זה.	

ROUNDS	כמות סבבי האימון שיבוצעו, כמפורט בתמצית רצף האימון תחת מבנה הפרויקט.	סה"כ יבוצעו $ROUNDS \times FITS$ איטרציות אימון. בכל איטרציית אימון הרשת תעשה $EPOCHS_PER_FIT$ אפוקים על $MINIBATCH\ SIZE$ חוויות.
FITS	איטרציות האימון בכל סבב.	
ENVS	כמות הסביבות המקבילות בהן יתבצע ייצור הנתונים.	
FIT FREQ	כמות איטרציות ייצור הנתונים על כל איטרציית אימון. במילים אחרות: כמות הצעדים הרצופים בכל אחת מהסביבות, שעבורם תתבצע איטרציית אימון אחת של הרשת.	
EPOCHS PER FIT	כמות האפוקים באיטרציית אימון אחת של הרשת ($model.fit$)	
MINIBATCH SIZE	כמות החוויות הנשלפות מהזיכרון בכל איטרציית אימון.	סך כל החוויות שייצברו הוא $ROUNDS \times FITS \times ENVS \times FIT\ FREQ$
MEMORY SIZE	גודל הזיכרון (ה- <i>experience buffer</i>) המקסימלי.	
MIN MEMORY SIZE	כמות החוויות המינימלית לתחילת אימון הרשת.	כשהזיכרון מלא, חוויות ישנות נדחפות החוצה ומפנות מקום לחוויות חדשות.
TARGET SYNC FREQ	כל כמה איטרציות לעדכן את ה- <i>target network</i> במשקולות של הרשת העיקרית.	
TAU	מקדם <i>soft update</i> , ראה תחת תיוג הנתונים.	$1 = \text{hard update}$
GAMMA	מקדם הנחה, ראה רקע תיאורטי.	
EPSILON	ערך בין 0 ל-1, הקובע את ההסתברות שבמהלך ייצור הנתונים תתבצע פעולה אקראית במקום זו שהסוכן בחר.	פעולה אקראית נקראת <i>exploration</i> , ופעולה של הסוכן נקראת <i>exploitation</i> . <i>exploration</i> חשוב על מנת לוודא שהסוכן חוקר הרבה אפשרויות, ולא "נתקע" על טקטיקה מסוימת שמובילה למקסימום מקומי של ביצועים.
EPSILON DECAY	אנו רוצים ש- <i>EPSILON</i> ידעך לאורך האימון משום שהסוכן כבר חקר הרבה אפשרויות והשתפר. הדעיכה <i>EPSILON DECAY</i> הוא בסיס החזקה.	המחשה לדעיכת <i>epsilon</i> לאורך האימון:
MIN EPSILON	ערך מינימלי בו <i>EPSILON</i> מפסיק לדעוך, על מנת לוודא שתמיד יהיה "מחקר" של טקטיקות נוספות.	
DECAY BEFORE TRAINING	האם להתחיל את דעיכת <i>EPSILON</i> במהלך ייצור הנתונים הראשוני, לפני תחילת אימון הרשת.	

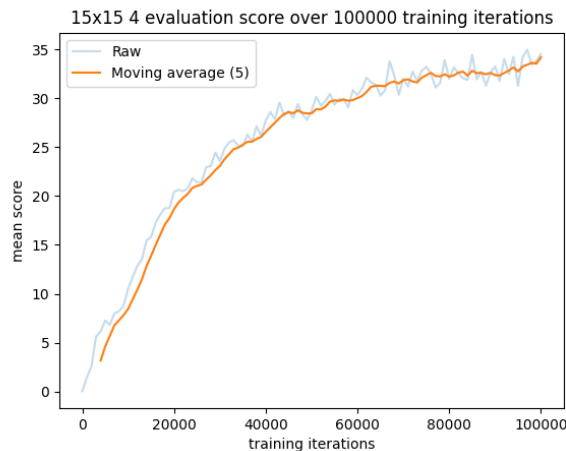
תוצאות אימון המודל בדוגמה

לאחר אימון ביצועי הסוכן עלו מ-0.03 ל-34.54 ממוצע תפוחים במשחק.

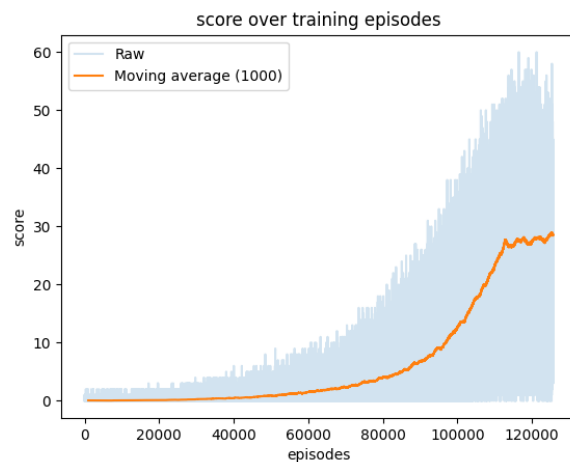
*לגרפים המוצגים בפרק זה ביצעתי החלקה באמצעות ממוצע נע ברוחב המצוין במקרא כל גרף.

ניתוח תוצאות ומדדי אימון: ממוצע התפוחים במשחקי אימון ומבחן

ממוצע תפוחים במשחקי מבחן



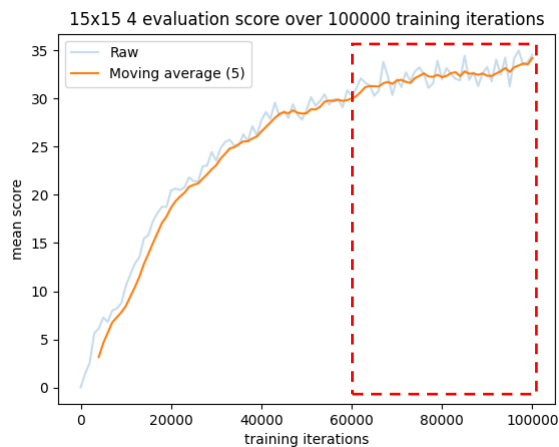
ממוצע תפוחים במשחקי אימון



במשחקי המבחן ניכר שיפור מהיר בביצועי המודל, בעוד שבמשחקי האימון ניכר שיפור מתון תחילה שלאחר מכן מאיץ. אני משער שתופעה זו נובעת מעצם העובדה שרק במהלך משחקי האימון מתבצע exploration, שעלול לגרום לפסילת הסוכן בטרם עת. כזכור, epsilon דועך אקספוננציאלית, לכן רוב ה-exploration מתבצע בתחילת האימון. בשלבי אימון מתקדמים epsilon נמוך ופחות מחבל בפעילות הסוכן. משחקי המבחן משקפים את היכולת האמיתית של המודל, משום שבהם אין exploration.

כעת נביט בגרפים לאחר שסימנתי עליהם מלבנים אדומים, המסמלים את אותו פרק אימון.

ממוצע תפוחים במשחקי מבחן



ממוצע תפוחים במשחקי אימון



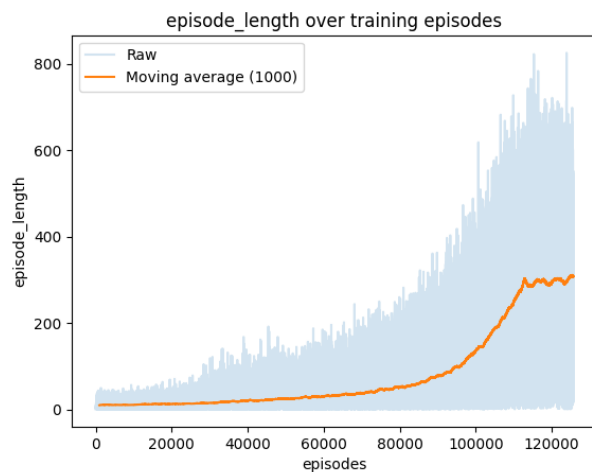
(הסבר בעמ' הבא)

ההסבר ל-"דחיסות" השונה נעוץ ביחידות ששימשו למדידת הביצועים באימון ובמבחן. ציר ה-x בגרף האימון נמדד במשחקים, בעוד שבגרף המבחן הוא נמדד באיטרציות אימון. במספר קבוע של איטרציות אימון ניתן להספיק מעט משחקים ארוכים או הרבה משחקים קצרים. משמע, משחקים ארוכים ידחסו את גרף האימון אך לא את גרף המבחן. ככל שהמודל משתפר הוא נפסל פחות, לכן המשחקים מתארכים והגרף נדחס. בנוסף, משחקים שטרם נגמרו בסיום האימון לא נכנסו לגרף האימון אך כן נספרו לגרף המבחן (מדובר ב-100 משחקים בלבד לכן השפעתם זניחה).

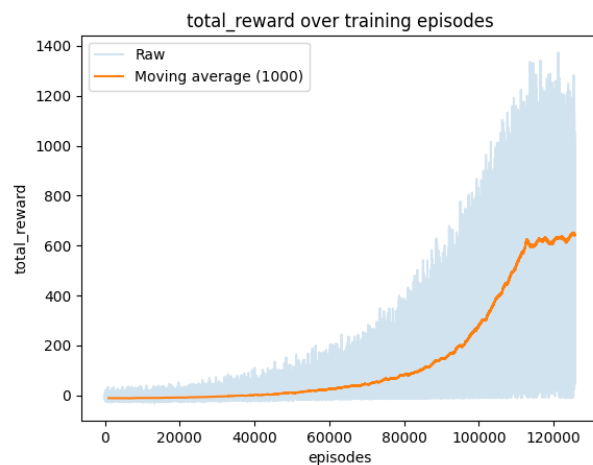
ניתוח תוצאות ומדדי אימון: אורך משחק וסכום תגמולים למשחק

הגרף הימני מתייחס לסכום התגמולים שנצברו בכל משחק עד לפסילה, והגרף השמאלי מתייחס לכמות הפעולות עד לפסילה.

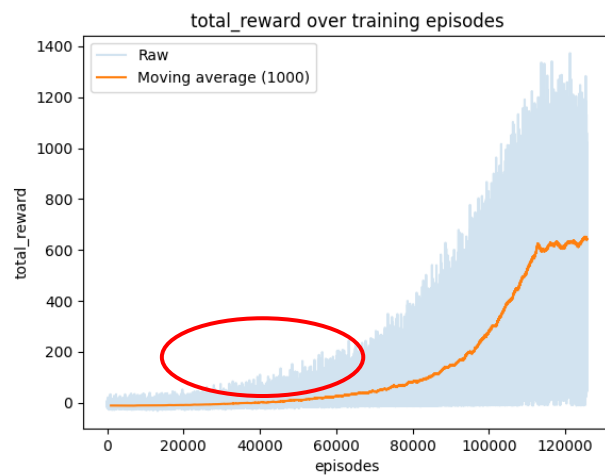
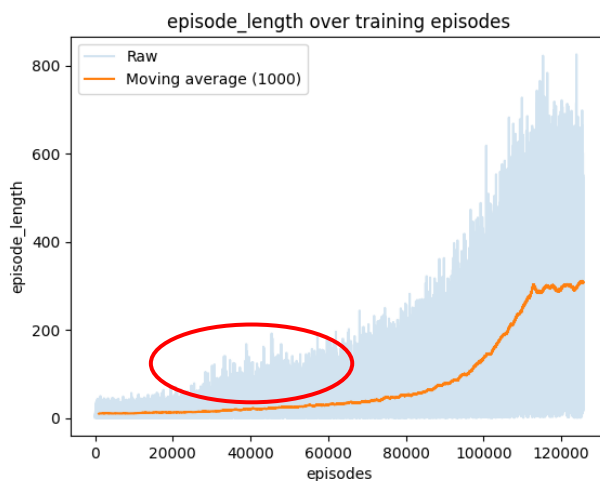
אורך משחקי אימון



סכום תגמולים למשחקי אימון



בגרף אורך המשחקים ניתן לראות "שפיצים" רבים שלא מופיעים בגרף סכום התגמולים, במיוחד באזורים המסומנים באדום:

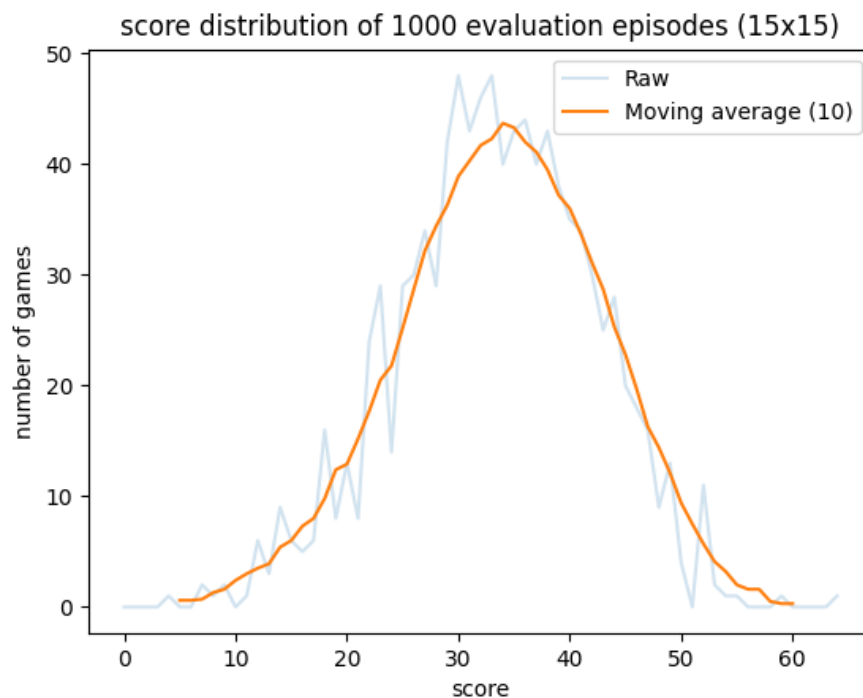


(הסבר בעמ' הבא)

אלו משחקים ארוכים במיוחד, אך עם ביצועים (פרסים מצטברים) סטנדרטיים. להערכתי, סביר כי מדובר במשחקים בהם המודל היה מספיק טוב בשביל להימנע מפסילה לזמן ממושך, אך לא מספיק טוב בשביל לאסוף תפוחים. במקרים מסוג זה ייתכנו "לופים" (ראה קטיעת לולאות).

ניתוח תוצאות ומדדי אימון: התפלגות ניקוד

ציר ה-x הוא כמות התפוחים שנאספו במשחק מבחן, וציר ה-y הוא כמות המשחקים (מתוך 1000) בהם הושגה תוצאה זו. לאחר החלקת הגרף עם ממוצע נע ברוחב 10, ניתן לראות שביצועי המודל מזכירים התפלגות נורמלית.



ייעול אימונים

```
profile: bool = False
multi_processing: bool = False
save_log: bool = True
save_metrics: bool = True
save_graph: bool = True
verbose: bool = True
summary_email: bool = True
live_emails: bool = False

MODEL_NAME = '15x15 4'
OLD_MODEL_NAME = ''
```

כוח המחשוב שברשותי מוגבל ביותר (הקוד רץ על המעבד של הלפטופ שלי, ללא האצה בעזרת GPU) לכן ייחסתי חשיבות רבה להאצת האימון עצמו, ולקטיעה מוקדמת של אימונים בקונפיגורציות שמתבררות כלא יעילות כבר במהלך האימון. על מנת לעשות זאת, לאורך הזמן הוספתי אפשרויות לבקרת והאצת האימונים, שהשליטה בהן מוצגת בתמונה. העמודים הקרובים מכילים פירוט על הכלים השונים.

Profiling

Profiling הוא כלי למדידת ביצועים של תוכנה בזמן הריצה, שעוזר לזהות חלקים שצורכים הרבה זמן חישוב. שימוש ב-profiling הוביל אותי לתובנות שאיפשרו לי לייעל את הקוד במקומות הנכונים ולהבדיל בין העיקר לתפל מבחינת השפעה על משך האימון. לדוגמה, בזכות profiling גיליתי שתחזית ערכי Q בשלב ייצור הנתונים דורשת הכי הרבה זמן, גילוי שבסופו של דבר הוביל אותי להוסיף סביבה וקטורית.

בתמונה מופיע חלק מדו"ח profile של אימון מינימלי לדוגמה.

734895994 function calls (680136976 primitive calls) in 1938.907 seconds

Ordered by: internal time

ncalls	tottime	percall	cumtime	percall	filename:lineno(function)
686876	662.156	0.001	664.024	0.001	{built-in method tensorflow.python._pywrap_tfe.TFE_Py_FastPathExecute}
55045	209.612	0.004	209.612	0.004	{built-in method tensorflow.python._pywrap_tfe.TFE_Py_Execute}
553617/15	178.895	0.000	0.012	0.001	C:\Users\yoavg\Github\Snake-DQN\venv\Lib\site-packages\tensorflow\python
571653	26.732	0.000	26.732	0.000	{built-in method tensorflow.python.util._tf_stack.extract_stack}
77	23.196	0.301	23.196	0.301	{method 'read' of '_ssl._SSLSocket' objects}
571651/3353	19.846	0.000	0.284	0.000	C:\Users\yoavg\Github\Snake-DQN\venv\Lib\site-packages\tensorflow\pyth
34883469/11189089	19.335	0.000	34.236	0.000	C:\Users\yoavg\Github\Snake-DQN\venv\Lib\site-packages\tensorflo
73524849/73524722	18.684	0.000	43.057	0.000	{built-in method builtins.isinstance}
77	15.269	0.198	15.269	0.198	{method 'write' of '_ssl._SSLSocket' objects}
173032	13.117	0.000	13.117	0.000	{method 'copy' of 'numpy.ndarray' objects}
12588860	12.866	0.000	28.354	0.000	C:\Users\yoavg\Github\Snake-DQN\venv\Lib\site-packages\tensorflow\python
30625867/30620806	12.155	0.000	38.745	0.000	{built-in method builtinsgetattr}
1401757	11.093	0.000	32.071	0.000	C:\Users\yoavg\AppData\Local\Programs\Python\Python312\Lib\inspect.py:236
1401932/1401757	10.649	0.000	48.477	0.000	C:\Users\yoavg\AppData\Local\Programs\Python\Python312\Lib\inspect.
427170	9.125	0.000	16.516	0.000	C:\Users\yoavg\Github\Snake-DQN\venv\Lib\site-packages\tensorflow\python
571653	8.968	0.000	8.968	0.000	{built-in method tensorflow.python.client._pywrap_tf_session.TF_FinishOp

Multi-Processing

בדרך כלל ייצור הנתונים (אינטראקציה עם הסביבה) ואימון המודל על חוויות מתבצע לסירוגין, משום שרמת המודל לאחר אימון משפיעה על ייצור הנתונים. אך מאחר שאני משתמש ב-experience replay אין מניעה לבצע אותם במקביל, לכן הוספתי אפשרות להשתמש בספרייה multiprocessing לשם כך. ההבדל במשך האימון לא היה משמעותי, וכאשר התחלתי להשתמש גם ב-pygame גיליתי שנוצר קונפליקט בשילוב הספריות TensorFlow, Pygame, Multiprocessing שמאט את זמן הריצה. לכן העדפתי להפסיק להשתמש באופציה זו.

מדדי אימון

הוספתי אפשרות לשמור מדדי אימון כגון ממוצע תפוחים, אורך משחקים וסכום תגמולים למשחק. לשם הנוחות הוספתי פונקציה לייצור הגרפים של המדדים בסיום האימון, שמוציגים בתוצאות אימון המודל לדוגמה. התנודתיות הגבוהה של המדדים והכמות העצומה של הנתונים מהם מקשים על פרשנותם בצורתם המקורית, לכן אני מחליק אותם באמצעות ממוצע נע שמראה בגרף את המגמה הכללית של כל מדד. שקלול הערכים נעשה באופן אחיד, ע"י מתן משקל שווה לכל הנקודות שנכנסות לממוצע.

Checkpoints

בסיום כל סבב אימונים ומבחן (round), הוספתי checkpoint שמאפשרת לעצור את האימון (ולהמשיך אותו מאותה נקודה אם צריך) בלי לאבד את ההתקדמות עד לשלב זה, ע"י שמירת המודל, מדדי האימון ומדדי המבחן. נוסף לכך, ב-checkpoint מודפסים למשתמש תוצאות המבחן והערכת זמן לסיום האימון, ונשלח עדכון באימייל עם נתוני המבחן ותחזית לזמן סיום האימון.

Emails

האימונים לוקחים זמן רב בו אני בדרך כלל לא ליד המחשב, לכן כאשר האימון נגמר או כשניתן לקטוע אותו באמצע בשל חוסר יעילות, אני "מבזבז זמן" כשאני לא מתחיל לאמן מודל נוסף או משנה פרמטרים וממשיך את האימון. על מנת לייעל את התהליך הוספתי אפשרות לשליחת אימייל סיכום אימון ואפשרות לשליחת עדכונים חיים באימייל. כאשר האפשרויות מופעלות, השליחה נעשית מהקוד באמצעות החבילות email, smtplib.

בעמוד הבא דוגמאות לשני סוגי האימיילים.

אימייל סיכום לדוגמה:

17x17 demo training summary python x

to me ▾

Wed, Apr 23, 8:43 AM (4 days ago)



2025-04-23 08:43:05

Finished training 17x17 demo

Training started at 2025-04-22 08:02:05 and finished at 2025-04-23 08:43:04 with a total runtime of 1 day, 0:40:59

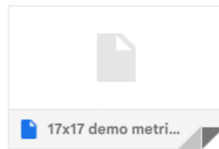
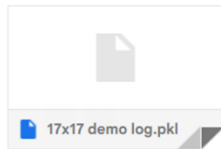
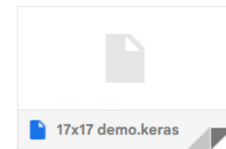
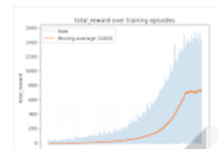
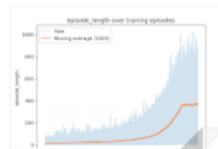
This email contains the following attachments:

- the trained model
- evaluation & training metrics
- graphs of the metrics

This email has been checked for viruses by Avast antivirus software.

www.avast.com

7 Attachments • Scanned by Gmail ⓘ



אימייל עדכון במהלך האימון:

round 18/100 live update - 17x17 demo log python x

to me ▾

Tue, Apr 22, 12:05 PM (7 days ago)



2025-04-22 12:05:34

Estimated finish: 2025-04-23 07:45:00

Raw evaluation data so far:

```
[["training iterations", "mean score", "17x17 demo performance over 100000 training iterations"], [[0, 1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000, 11000, 12000, 13000, 14000, 15000, 16000, 17000, 18000], [0.01, 2.47, 4.92, 6.0, 6.71, 6.92, 7.9, 7.87, 9.21, 10.23, 11.15, 11.37, 12.5, 13.33, 15.85, 15.42, 17.21, 18.78, 19.45], '17x17 demo']]
```

This email has been checked for viruses by Avast antivirus software.

www.avast.com

One attachment • Scanned by Gmail ⓘ



Verbose

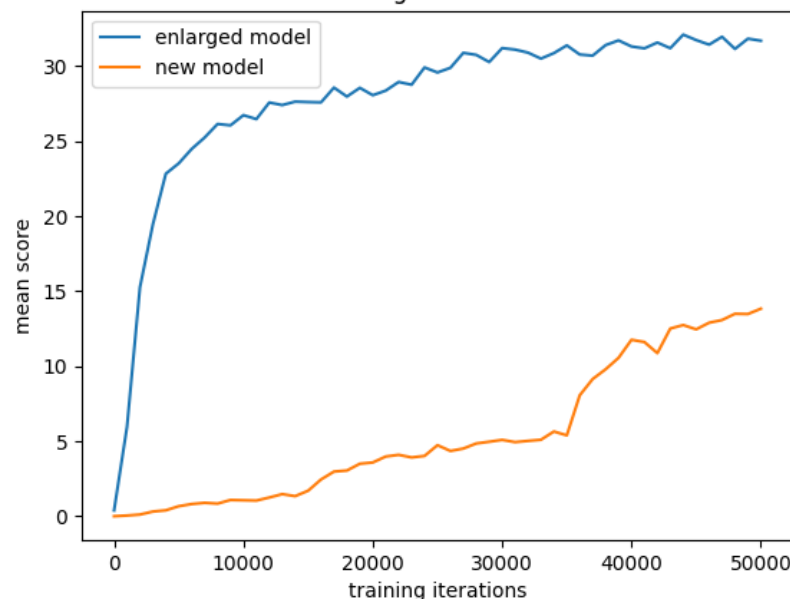
לשם בקרה על האימון הוספתי פלט מפורט המודפס על המסך, הכולל מידע מועיל כגון סטטוס האימון/המבחן, ביצועי הסוכן, והערכת זמן לסיום האימון. דוגמה לפלט מתחילת האימון:

```
2025-04-22 08:02:05 : starts evaluating round 0 / 100
2025-04-22 08:02:05 : starts gathering initial experience
2025-04-22 08:02:54 : finished gathering initial experience
2025-04-22 08:02:54 : starts 100000 training iterations. evaluation every 1000
2025-04-22 08:02:54 : starts training round 1 / 100
2025-04-22 08:13:27 : starts evaluating round 1 / 100
2025-04-22 08:13:44 : evaluation result: 2.47 estimated finish: 2025-04-23 02:06:14
Message sent successfully!
2025-04-22 08:13:46 : starts training round 2 / 100
2025-04-22 08:24:32 : starts evaluating round 2 / 100
2025-04-22 08:24:51 : evaluation result: 4.92 estimated finish: 2025-04-23 02:31:01
Message sent successfully!
2025-04-22 08:24:53 : starts training round 3 / 100
2025-04-22 08:36:01 : starts evaluating round 3 / 100
2025-04-22 08:36:23 : evaluation result: 6.0 estimated finish: 2025-04-23 03:11:53
Message sent successfully!
```

הגדלת מודל ואימון מקדים

התנסיתי בלאמן מודלים על לוחות בגדלים שונים, וסיקרן אותי לראות אם למודל שאומן על לוח קטן יהיה יתרון בהתכנסות על לוח גדול יותר. האופן המדויק בו מתבצע המעבר מפורט תחת **בניית המודל**, אך עיקר הבעיה הוא שקלט גדול יותר לרשת דורש משקולות נוספות. בהתחלה ניסיתי להוסיף את המשקולות הנדרשות באמצעות אינטרפולציה לינארית, אך התוצאה היתה שהמודל ננעל על כיוון אחד ומתקדם כמעט ורק אליו, ללא תלות בקלט. לאחר מכן ניסיתי לרפד באפסים את המשקולות החסרות, והתוצאות היו מדהימות:

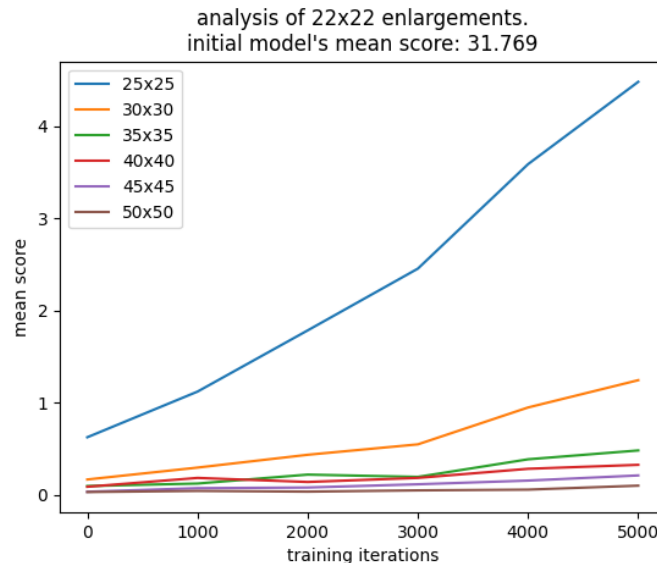
Evaluations Mean Score throughout Training of Enlarged vs New Model.
both are 22x22 models. enlargement from a trained 18x18 model.



בעקבות ההצלחה של הגדלת מודל, עלתה לי השאלה:

עד כמה אפשר להגדיל מודל בלי לאבד את יעילות ההגדלה?

על מנת לענות על שאלה זו ניסיתי לבצע אימון מקדים הכולל פרק אימון קצר למספר רב של הגדלות למודל מסוים, ולפיו לנסות לחזות את היעילות של המשך האימונים. השתמשתי במודל מאומן בגודל 22x22 והגדלה בקפיצות של 5, ואלו התוצאות שקיבלתי:



בגרף ניכר כי במודל 30x30 ניכר שיפור יחסית מהיר, בעוד במודל 50x50 בקושי ניכר שיפור בפרק האימון המקדים. המשכתי רק את האימון של שני המודלים הללו (על מנת לחסוך במשאבי מחשב), והגעתי לתוצאה הבאה:



לצערי נאלצתי להפסיק את האימונים בשל העלות החישובית הגבוהה שלהם (במיוחד בלוחות מאוד גדולים) ובשל משאבי המחשב הדלים שברשותי. להערכתי לא ניתן להסיק מסקנות חד משמעיות מהמידע שנצבר ללא המשך ניסויים.

יישום

יישום המודל הוא להפעיל אותו בסביבה, כלומר להשתמש בו לשחק Snake.

המשתמש בוחר הגדרות כגון שם המודל, אורך הנחש, כמות התפוחים, קטיעת לולאות, fps, כמות משחקים וכו' שמפורטים במדריך למשתמש.

ניתן לבחור ביישום ויזואלי של המודל המציג את משחק המודל באמצעות הספרייה Pygame, או לבחור ביישום המודל ללא אנימציה, אשר מריץ את המשחקים במקביל לשם יעילות מרבית. המשחק הטוב ביותר נשמר וניתן לצפות בו מחדש.

הפונקציה המשמשת ליישום לא ויזואלי היא אותה פונקציה המשמשת למבחן בזמן האימון, אך היישום הוויזואלי נעשה באמצעות פונקציה אחרת הדומה במבנה שלה.

משחקים לדוגמה של מודל ה-15x15 מהדוגמה בבניית המודל:

לפני אימון:

<https://vimeo.com/1081900645/b503552aeb>

אחרי אימון:

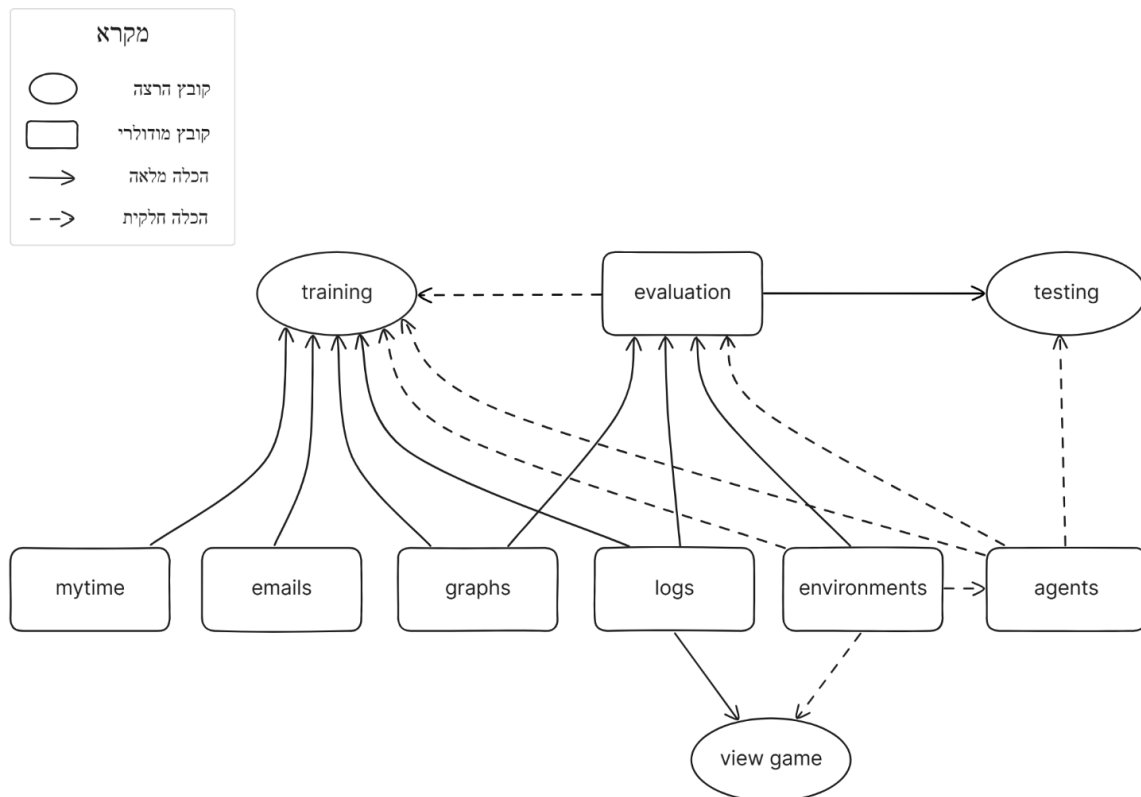
<https://vimeo.com/1081900675/fde202df00>

מדריך למפתח

הפרויקט כתוב ב-Python 3.12. אני עבדתי ב-PyCharm Community Edition ואת רוב הספריות הורדתי דרך pip באופן המובנה ב-IDE. עבור עבודה בסביבות אחרות אפשר להתקין ספריות ישירות מ-pip.

קבצי הקוד בפרויקט מחולקים בצורה מודולרית. להבנה מיטבית, תחילה אסביר על המודולים ולאחר מכן על קבצי ההרצה (שמשתמשים במודולים). להלן סיכום תמציתי של קבצי הפרויקט:

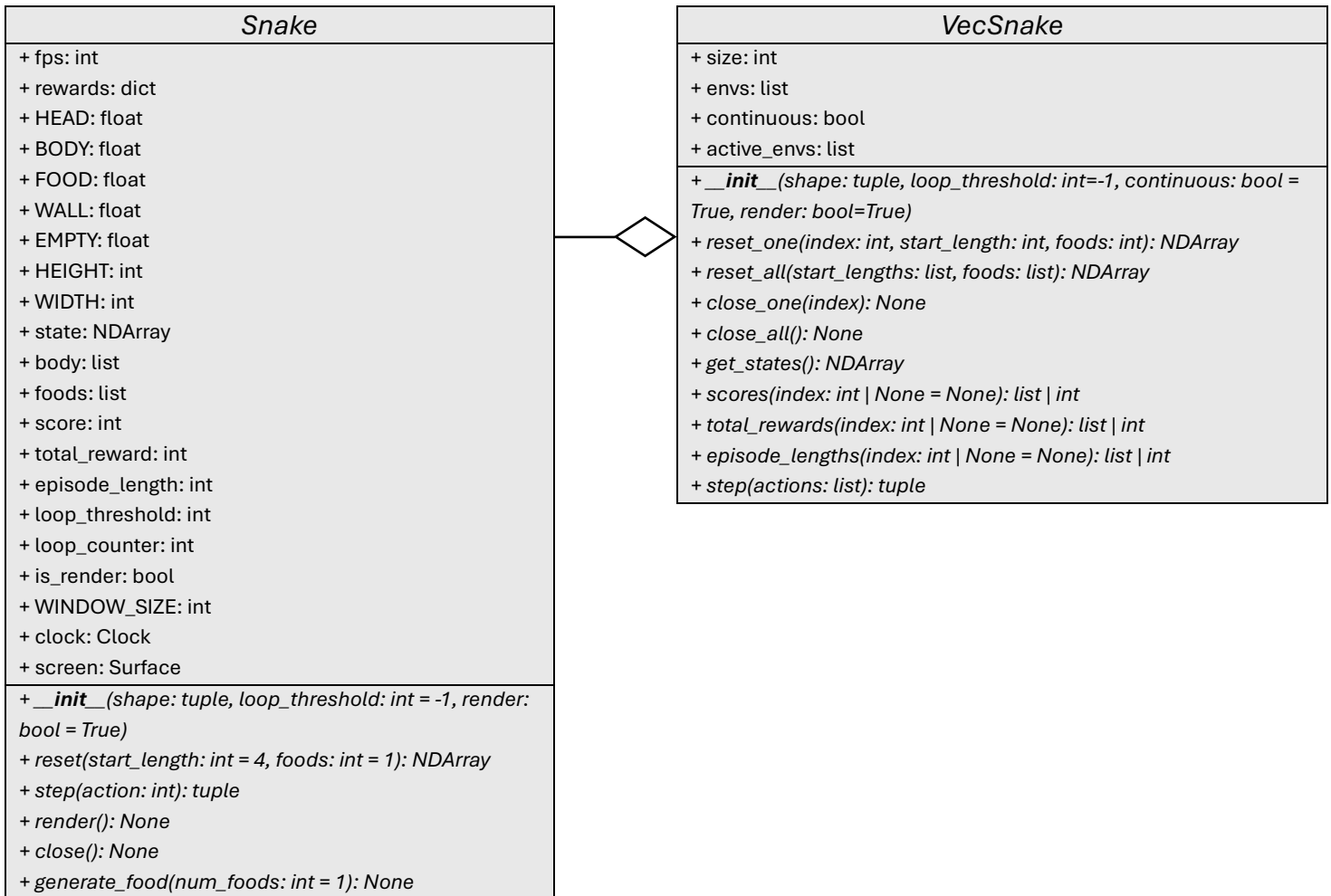
שם הקובץ (.py)	תיאור
קבצי הרצה	
<i>training</i>	קובץ האימון הראשי
<i>testing</i>	קובץ היישום הראשי
<i>view game</i>	צפייה במשחק שמור
מודולים	
<i>environments</i>	מחלקות הסביבה המשמשות לאימון ולמבחן
<i>agents</i>	מחלקת הסוכן, פעולות עזר ומחלקות עזר ליישום
<i>evaluation</i>	פונקציות מבחן ויזואלי ולא ויזואלי
<i>logs</i>	פונקציות לשמירת וקריאת נתוני אימון ומבחן
<i>graphs</i>	פונקציות לייצור גרפים
<i>emails</i>	פונקציה לשליחת אימייל
<i>mytime</i>	פונקציות להערכת זמן ריצה



environments.py

מודול המכיל את המחלקות Snake, VecSnake, Snake3, VecSnake3. הסיומת 3 מתייחסת לתנועה יחסית (3 פעולות), והמחלקות ללא סיומת מיישמות תנועה מוחלטת (4 פעולות). הקידומת Vec מעידה על כך שזו סביבה וקטורית. למחלקות מבנה דומה לכן אציג רק את חלקן.

תרשים UML של המחלקות Snake ו-VecSnake (קישורים לפעולות בלחיצה):



ייבוא ספריות לקובץ environments:

```

import random
import numpy as np
from numpy.typing import NDArray
import os

os.environ['PYGAME_HIDE_SUPPORT_PROMPT'] = '1' # Hide pygame's welcome output
import pygame
  
```

Snake – הגדרת המחלקה ותכונות סטטיות:

```
class Snake:
    fps = 30 # fps limit; only relevant when rendering
    rewards = {'food': 15, 'hit wall': -10, 'hit body': -10, 'survive': 0, 'move closer': 1, 'move further': -1}
    # move closer and move further rewards only apply if MAX_FOOD=1
    # representations for different tile types:
    HEAD = 0.5
    BODY = -0.5
    FOOD = 1
    WALL = -1
    EMPTY = 0
```

Snake.__init__ – פעולה בונה:

```
def __init__(self, shape: tuple[int, int], loop_threshold: int = -1, render: bool = True):
    """
    :param loop_threshold: negative value = no loop declaration
    """
    # declaring variables
    self.HEIGHT, self.WIDTH = shape
    self.state: NDArray[float] = np.array([])
    self.body: list[tuple[int, int]] = []
    self.foods: list[tuple[int, int]] = []
    # metrics
    self.score: int = 0
    self.total_reward: int = 0
    self.episode_length: int = 0
    # loop detection
    self.loop_threshold: int = loop_threshold # consecutive moves without food before declaring loop
    self.loop_counter: int = 0
    # rendering
    self.is_render: bool = render
    if self.is_render:
        self.clock = pygame.time.Clock()
        self.TILE_SIZE = 750 // self.HEIGHT
        pygame.init()
        self.screen = pygame.display.set_mode((self.HEIGHT * self.TILE_SIZE, self.WIDTH * self.TILE_SIZE))
        pygame.display.set_caption('Snake')
```

Snake.reset – אתחול המשחק:

```

def reset(self, start_length: int = 4, foods: int = 1) -> NDArray[float]:
    # Reset variables
    self.loop_counter = 0
    self.body = []
    self.foods = []
    self.score = 0
    self.total_reward = 0
    self.episode_length = 0
    self.state = np.zeros(shape=(self.HEIGHT, self.WIDTH))
    self.state[0] += self.WALL
    self.state[-1] += self.WALL
    for i in range(1, self.HEIGHT - 1):
        self.state[i][0] = self.WALL
        self.state[i][-1] = self.WALL

    # Choose random starting position and direction
    start_pos = None
    look_dir = random.randint(1, 4)
    match look_dir:
        case 1: # Up
            start_pos = (random.randint(2, self.HEIGHT - start_length - 2),
                        random.randint(2, self.WIDTH - 3))
            for i in range(1, start_length):
                self.state[start_pos[0] + i, start_pos[1]] = self.BODY
                self.body.insert(0, (start_pos[0] + i, start_pos[1]))
        case 2: # Right
            start_pos = (random.randint(2, self.HEIGHT - 3),
                        random.randint(start_length + 1, self.WIDTH - 2))
            for i in range(1, start_length):
                self.state[start_pos[0], start_pos[1] - i] = self.BODY
                self.body.insert(0, (start_pos[0], start_pos[1] - i))
        case 3: # Down
            start_pos = (random.randint(start_length + 1, self.HEIGHT - 2),
                        random.randint(2, self.WIDTH - 3))
            for i in range(1, start_length):
                self.state[start_pos[0] - i, start_pos[1]] = self.BODY
                self.body.insert(0, (start_pos[0] - i, start_pos[1]))
        case 4: # Left
            start_pos = (random.randint(2, self.HEIGHT - 3),
                        random.randint(2, self.WIDTH - start_length - 2))
            for i in range(1, start_length):
                self.state[start_pos[0], start_pos[1] + i] = self.BODY
                self.body.insert(0, (start_pos[0], start_pos[1] + i))

    # Add head
    self.body.append(start_pos)
    self.state[start_pos[0], start_pos[1]] = self.HEAD

    self.generate_food(foods)
    if self.is_render: self.render()
    return np.copy(self.state)

```

Snake.step – צעד בסביבה:

```

def step(self, action: int) -> tuple[NDArray[float], int, bool]:
    done = False # Assign default value
    # Update the snake's body based on action
    match action:
        case 0: # Up
            if self.body[-1][0] - 1 != self.body[-2][0]:
                self.body.append((self.body[-1][0] - 1, self.body[-1][1]))
            else:
                self.body.append((self.body[-1][0] + 1, self.body[-1][1]))
        case 1: # Right
            if self.body[-1][1] + 1 != self.body[-2][1]:
                self.body.append((self.body[-1][0], self.body[-1][1] + 1))
            else:
                self.body.append((self.body[-1][0], self.body[-1][1] - 1))
        case 2: # Down
            if self.body[-1][0] + 1 != self.body[-2][0]:
                self.body.append((self.body[-1][0] + 1, self.body[-1][1]))
            else:
                self.body.append((self.body[-1][0] - 1, self.body[-1][1]))
        case 3: # Left
            if self.body[-1][1] - 1 != self.body[-2][1]:
                self.body.append((self.body[-1][0], self.body[-1][1] - 1))
            else:
                self.body.append((self.body[-1][0], self.body[-1][1] + 1))

    # Determine reward, detect if done, and update state
    match self.state[self.body[-1][0], self.body[-1][1]]:
        case self.EMPTY:
            # Check if snake moved closer/further from the food
            # Only applies with 1 food on the board
            if len(self.foods) == 1:
                diff = (np.sqrt((self.body[-2][0] - self.foods[0][0]) ** 2 + (self.body[-2][1] -
self.foods[0][1]) ** 2) - np.sqrt((self.body[-1][0] - self.foods[0][0]) ** 2 + (self.body[-1][1]
- self.foods[0][1]) ** 2))
                if diff > 0:
                    reward = self.rewards['move closer']
                elif diff < 0:
                    reward = self.rewards['move further']
                else:
                    reward = self.rewards['survive']
            else:
                reward = self.rewards['survive']
        # Update state
        self.state[self.body[-1][0], self.body[-1][1]] = self.HEAD
        self.state[self.body[-2][0], self.body[-2][1]] = self.BODY
        self.state[self.body[0][0], self.body[0][1]] = self.EMPTY
        self.body.pop(0)
        # Apply loop detection
        self.loop_counter += 1
        if self.loop_counter == self.loop_threshold:
            done = True
    case self.FOOD:

```

```

        self.loop_counter = 0 # Reset loop detection
        reward = self.rewards['food']
        # Update state
        self.state[self.body[-1][0], self.body[-1][1]] = self.HEAD
        self.state[self.body[-2][0], self.body[-2][1]] = self.BODY
        # Update foods
        self.foods.remove(self.body[-1])
        self.generate_food(1)
        self.score += 1
    case self.BODY:
        reward = self.rewards['hit body']
        done = True
    case self.WALL:
        reward = self.rewards['hit wall']
        done = True
    case _:
        raise ValueError('Invalid tile value reached')

# Update metrics
self.episode_length += 1
self.total_reward += reward
# Update animation
if self.is_render:
    self.render()
    self.clock.tick(self.fps)
return np.copy(self.state), reward, done # return feedback

```

Snake.render – עדכון האנימציה:

```

def render(self):
    # Iterate over tiles
    for y in range(self.HEIGHT):
        for x in range(self.WIDTH):
            rect = pygame.Rect(x * self.TILE_SIZE, y * self.TILE_SIZE, self.TILE_SIZE, self.TILE_SIZE)
            # Draw the tile and it's outline
            match self.state[y, x]:
                case self.EMPTY:
                    pygame.draw.rect(self.screen, (0, 150, 0), rect)
                    pygame.draw.rect(self.screen, (0, 175, 0), rect, 1)
                case self.WALL:
                    pygame.draw.rect(self.screen, (128, 128, 128), rect)
                case self.BODY:
                    pygame.draw.rect(self.screen, (0, 0, 125), rect)
                    pygame.draw.rect(self.screen, (0, 0, 0), rect, 1)
                case self.HEAD:
                    pygame.draw.rect(self.screen, (0, 0, 255), rect)
                    pygame.draw.rect(self.screen, (0, 0, 0), rect, 1)
                case self.FOOD:
                    pygame.draw.rect(self.screen, (255, 0, 0), rect)
                    pygame.draw.rect(self.screen, (0, 0, 0), rect, 1)
    pygame.display.flip() # Update the display

```

Snake.close – סגירת האנימציה:

```
def close(self):
    if self.is_render:
        pygame.quit()
```

Snake.generate_food – יצירת אוכל חדש:

```
def generate_food(self, num_foods: int = 1) -> None:
    for i in range(num_foods): # Repeat for the amount of required foods
        # Choose a random empty tile
        pos = random.randint(0, self.HEIGHT - 1), random.randint(0, self.WIDTH - 1)
        while self.state[pos[0], pos[1]] != self.EMPTY:
            pos = random.randint(0, self.HEIGHT - 1), random.randint(0, self.WIDTH - 1)
        # Assign tile as food
        self.state[pos[0], pos[1]] = self.FOOD
        self.foods.append(pos)
```

VecSnake.__init__ – פעולה בונה:

```
class VecSnake:
    def __init__(self, shape: tuple[int, int, int], loop_threshold: int = -1, continuous: bool = True,
render: bool = True):
        """
        :param shape: (height, width, envs)
        :param loop_threshold: negative value = no loop declaration
        :param continuous: whether to reset done envs without waiting for all
        envs
        :param render:
        """
        self.size = shape[2]
        self.envs = [Snake((shape[0], shape[1]), loop_threshold, render) for i in range(self.size)]
        self.continuous = continuous
        if not self.continuous:
            self.active_envs = self.envs.copy()
```

VecSnake.reset_one – איפוס סביבה אחת:

```
def reset_one(self, index: int, start_length: int, foods: int) -> NDArray[float]:
    return self.envs[index].reset(start_length, foods)
```

VecSnake.reset_all – איפוס כל הסביבות:

```
def reset_all(self, start_lengths: list[int], foods: list[int]) -> NDArray[float]:
    return np.array([self.envs[i].reset(start_lengths[i], foods[i]) for i in range(self.size)])
```

VecSnake.close_one – סגירת סביבה אחת:

```
def close_one(self, index):
    if self.continuous:
        self.envs[index].close()
    else:
        self.active_envs[index].close()
```

VecSnake.close_all – סגירת כל הסביבות:

```
def close_all(self):
    for i in range(self.size):
        self.envs[i].close()
```

VecSnake.get_states – אחזור המצבים בלוחות:

```
def get_states(self) -> NDArray[float]:
    return np.array([np.copy(self.envs[i].state) for i in range(self.size)])
```

VecSnake.scores – אחזור מדד הניקוד:

```
def scores(self, index: int | None = None) -> list[int] | int:
    if index is None:
        return [env.score for env in self.envs]
    return self.envs[index].score
```

VecSnake.total_rewards – אחזור מדד התגמולים המצטברים:

```
def total_rewards(self, index: int | None = None) -> list[int] | int:
    if index is None:
        return [env.total_reward for env in self.envs]
    return self.envs[index].total_reward
```

VecSnake.episode_lengths – אחזור מדד אורך המשחק:

```
def episode_lengths(self, index: int | None = None) -> list[int] | int:
    if index is None:
        return [env.episode_length for env in self.envs]
    return self.envs[index].episode_length
```


VecSnake.step – צעד בסביבות:

```
def step(self, actions: list[int]) -> tuple[list[NDArray[float]], list[int], list[bool]]:
    # Initialize feedbacks
    new_states = []
    rewards = []
    done = []
    # Iterate over all active environments
    for i in range(len(actions)):
        # Perform step
        if self.continuous:
            feedback = self.envs[i].step(actions[i])
        else:
            feedback = self.active_envs[i].step(actions[i])
        # Save feedback
        new_states.append(feedback[0])
        rewards.append(feedback[1])
        done.append(feedback[2])

    return new_states, rewards, done # Return feedbacks
```

agents.py

מודול המכיל את המחלקות *RandomPlayer*, *HumanPlayer*, *DQNAgent*

מכיל את הפונקציות *mask_loss*, *unpack_tensor_reshape*, *tensor_reshape*

תרשים UML של המחלקות (קישורים לפעולות בלחיצה):

<i>HumanPlayer</i>
+ input_shape: tuple
+ action: int
+ __init__ (input_shape: tuple)
+ predict(state=None, verbose=None): list

<i>RandomPlayer</i>
+ input_shape: tuple
+ __init__ (input_shape: tuple)
+ predict(current_state=None, verbose=None): list

<i>DQNAgent</i>
+ save_name: str
+ model: keras.Model
+ target_model: keras.Model
+ target_sync_counter: int
+ memory: deque
+ __init__ (save_name: str, memory_size: int, shape: tuple, old_model=None)
+ create_model(new_shape: tuple, old_model=None, action_space: int = 4): keras.Model
+ fit_minibatch(minibatch_size: int, epochs: int, gamma: float, tau: float, target_sync_freq: int): None
+ update_memory(experiences: list): None
+ save(save_name: str = None): None

ייבוא ספריות לקובץ agents:

```
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # Reduce tensorflow's verbose
os.environ['PYGAME_HIDE_SUPPORT_PROMPT'] = '1' # Hide pygame's welcome output
import tensorflow as tf
from tensorflow.keras.layers import Input, Flatten, Dense, Conv2D
from environments import Snake
import numpy as np
from numpy.typing import NDArray
import random
from collections import deque
import pygame
```

DQNAgent.__init__ – פעולה בונה:

```
class DQNAgent:
    def __init__(self, save_name: str, memory_size: int, shape: tuple[int, int], old_model=None):
        # Creation of a new model if it doesn't exist already
        if not os.path.exists(save_name):
            self.create_model(shape, old_model).save(save_name)
        self.save_name = save_name
```

```

# Load model
self.model = tf.keras.models.load_model(save_name)
# Initialize target network
self.target_model = tf.keras.models.clone_model(self.model)
self.target_model.set_weights(self.model.get_weights())
self.target_sync_counter = 0
# Initialize experience buffer
self.memory = deque(maxlen=memory_size)

```

DQNAgent.create_model – יצירת מודל חדש:

```

def create_model(self, new_shape: tuple[int, int], old_model=None, action_space: int = 4):
    """
    :param action_space: Output shape; either 3 or 4
    :param new_shape: New model's input shape
    :param old_model: If passed, weights will be transferred.
    :return: A new model
    """

    # New model definition. Should be identical to old model except input layer.
    inputs = Input(shape=(new_shape[0], new_shape[1], 1))
    x = Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(inputs)
    x = Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(x)
    x = Flatten()(x)
    x = Dense(128, activation='relu')(x)
    value = Dense(1, activation='linear')(x)
    advantage = Dense(action_space, activation='linear')(x)
    mean_advantage = tf.keras.ops.mean(advantage, axis=1, keepdims=True)
    q_values = value + (advantage - mean_advantage)

    new_model = tf.keras.Model(inputs=inputs, outputs=q_values)

    # Transfer weights if given
    if old_model is not None:
        for old_layer, new_layer in zip(old_model.layers, new_model.layers):
            if isinstance(new_layer, Dense):
                # Zero padding to the end of the array.
                old_weights, old_biases = old_layer.get_weights()
                pad_size = new_layer.get_weights()[0].shape[0] - old_weights.shape[0]
                padded_weights = np.pad(old_weights, ((0, pad_size), (0, 0)),
                                         mode='constant', constant_values=0)
                new_layer.set_weights([padded_weights, old_biases])
            elif isinstance(new_layer, Conv2D):
                # No padding
                new_layer.set_weights(old_layer.get_weights())

    new_model.compile(loss='mean_squared_error', optimizer='adam')
    return new_model

```

DQNAgent.fit_minibatch – איטרציית אימון לרשת:

```
def fit_minibatch(self, minibatch_size: int, epochs: int, gamma: float, tau: float, target_sync_freq: int) -> None:
    minibatch = random.sample(self.memory, minibatch_size) # Retrieve experiences
    # Predict Q(s, a)
    current_states = tensor_reshape(np.array([experience[0] for experience in minibatch]), is_batch=True)
    current_qs_minibatch = self.model.predict(current_states, verbose=0)
    # Predict Q(s', a')
    new_states = tensor_reshape(np.array([experience[3] for experience in minibatch]), is_batch=True)
    future_qs_minibatch = self.target_model.predict(new_states, verbose=0)
    # Initialize data structures
    x = []
    y = []
    # Label data
    for index, (current_state, action, reward, new_state, done) in enumerate(minibatch):
        # Calculate Q(s, a) using Bellman equation
        if done:
            new_q = reward
        else:
            new_q = reward + gamma * np.max(future_qs_minibatch[index])
        # Combine prediction & calculation to create y labels
        current_qs = current_qs_minibatch[index]
        current_qs[action] = new_q
        # Append to training dataset
        x.append(current_state)
        y.append(current_qs)
    # Train the model
    self.model.fit(np.array(x), np.array(y), epochs=epochs, batch_size=minibatch_size, verbose=0, shuffle=False)
    # Synchronize target network if needed
    self.target_sync_counter += 1
    if self.target_sync_counter >= target_sync_freq:
        model_weights = np.array(self.model.get_weights(), dtype=object)
        target_weights = np.array(self.target_model.get_weights(), dtype=object)
        new_weights = tau * model_weights + (1.0 - tau) * target_weights
        self.target_model.set_weights(new_weights)
        self.target_sync_counter = 0
```

DQNAgent.update_memory – הוספת חוויות ל-*experience buffer*:

```
def update_memory(self, experiences: list[tuple]) -> None:
    self.memory.extend(experiences)
```

DQNAgent.save – שמירת המודל:

```
def save(self, save_name: str = None):
    if save_name is None:
        self.model.save(self.save_name)
    else:
        self.model.save(save_name)
```

מחלקת HumanPlayer משמשת רק עבור משחקי מבחן של שחקן אנושי, ומממשת חלק מהפעולות של מודל keras בגרסה שמאפשרת קלט אנושי.

HumanPlayer.__init__ – פעולה בונה:

```
class HumanPlayer:
    def __init__(self, input_shape: tuple[int, int, int, int]):
        self.input_shape = input_shape
        self.action = 0
```

HumanPlayer.predict – קליטת פעולה ממשתמש אנושי:

```
def predict(self, state=None, verbose=None) -> list[NDArray[float]]:
    # Receive human input
    events = pygame.event.get()
    for event in events:
        if event.type == pygame.KEYDOWN:
            key = event.key
            if key == pygame.K_UP:
                self.action = 0
            if key == pygame.K_RIGHT:
                self.action = 1
            if key == pygame.K_DOWN:
                self.action = 2
            if key == pygame.K_LEFT:
                self.action = 3
    # Imitate keras output structure
    q_values = np.zeros(4)
    q_values[self.action] = 1
    return [q_values]
```

מחלקת RandomPlayer זהה ל-HumanPlayer בתפקידה ומימושה, מלבד זאת שהפלט שלה אקראי ולא קלט מהמשתמש.

RandomPlayer.__init__ – פעולה בונה:

```
class RandomPlayer:
    def __init__(self, input_shape: tuple[int, int, int, int]):
        self.input_shape = input_shape
```

RandomPlayer.predict – יצירת ערכי Q רנדומליים:

```
def predict(self, current_state=None, verbose=None) -> list[NDArray[float]]:
    # Imitate keras output structure with random values, for each parallel game
    q_values = [np.array([random.random() for action in range(4)]) for env in range(self.input_shape[0])]
    return q_values
```

פעולות עזר בקובץ agents:

tensor_reshape – הכנת הקלט לרשת:

```
def tensor_reshape(obs: NDArray[float], is_batch: bool = False) -> NDArray[float]:
    """
    Prepares observations for .predict
    :param obs: A single state or a batch of states
    :param is_batch: Whether a batch or a single state were given
    :return: Observations ready for .predict
    """
    if is_batch:
        return obs.reshape((obs.shape[0], obs.shape[1], obs.shape[2], 1)) # vectorized reshape
    return obs.reshape((1, obs.shape[0], obs.shape[1], 1)) # single state reshape
```

unpack_tensor_reshape – ההמרה ההפוכה מ-tensor_reshape:

```
def unpack_tensor_reshape(obs: NDArray[float], is_batch: bool = False) -> NDArray[float]:
    """
    Undoes the function tensor_reshape
    :param obs: A single state or a batch of states
    :param is_batch: Whether a batch or a single state were given
    :return: Observations in their regular form
    """
    if is_batch:
        return obs.reshape((obs.shape[0], obs.shape[1], obs.shape[2])) # vectorized reshape
    return obs.reshape((obs.shape[1], obs.shape[2])) # single state reshape
```

mask_loss – מניעת פסילה מיידיית (בפעולה הבאה בלבד):

```
def mask_loss(state: NDArray[float], body: list, qs: NDArray[float] | list[float]):
    # Prepare variables
    head_i, head_j = body[-1]
    qs = qs[0]
    old_action = np.argmax(qs)
    mask = min(qs) - 1
    # It's impossible to move backwards, and trying so results in forward movement.
    # Therefore, we mask every direction with an obstacle, unless the snake came from it:
    # Up
    if state[head_i - 1, head_j] == Snake.BODY or state[head_i - 1, head_j] == Snake.WALL and
head_i - 1 != body[-2][0]:
        qs[0] = mask
    # Right
    if state[head_i, head_j + 1] == Snake.BODY or state[head_i, head_j + 1] == Snake.WALL and
head_j + 1 != body[-2][1]:
        qs[1] = mask
    # Down
    if state[head_i + 1, head_j] == Snake.BODY or state[head_i + 1, head_j] == Snake.WALL and
head_i + 1 != body[-2][0]:
        qs[2] = mask
    # Left
    if state[head_i, head_j - 1] == Snake.BODY or state[head_i, head_j - 1] == Snake.WALL and
head_j - 1 != body[-2][1]:
        qs[3] = mask
    # And if there's an obstacle straight ahead, mask the backwards direction as well:
    if head_i - 1 == body[-2][0] and qs[2] == mask: qs[0] = mask
    if head_j + 1 == body[-2][1] and qs[3] == mask: qs[1] = mask
    if head_i + 1 == body[-2][0] and qs[0] == mask: qs[2] = mask
    if head_j - 1 == body[-2][1] and qs[1] == mask: qs[3] = mask
    # Notify user if masking changed the chosen action
    if np.argmax(qs) != old_action:
        print('used mask')
    return qs
```

evaluation.py

מודול המכיל את הפונקציות `visual_evaluation`, `score_evaluation`.

ייבוא ספריות לקובץ:

```

import graphs
import logs
import numpy as np
import random
from environments import VecSnake, Snake, Snake3, VecSnake3
from agents import tensor_reshape, DQNAgent, HumanPlayer, mask_loss
from time import sleep
import os

os.environ['PYGAME_HIDE_SUPPORT_PROMPT'] = '1'
import pygame

```

`visual_evaluation` – מבחן עם אנימציה. משתמש אנושי יכול לשחק באמצעות פונקציה זו. ללא סביבה וקטורית; המשחקים מתבצעים אחד אחרי השני.

```

def visual_evaluation(model, episodes: int = -1, loop_threshold: int = 200, foods: tuple[int, int] = (1, 1),
    start_lengths: tuple[int, int] = (4, 4), fps: int = 15, save_best: bool = True,
    old_actions: bool = False, mask: bool = False) -> float:
    """
    :param model: Either a Q-network or an agent
    :param episodes: -1 for endless evaluation
    :param loop_threshold: Negative value for no loop detection
    :param foods: (min foods, max foods). A random int in this range will be chosen each episode
    :param start_lengths: (min start length, max start length). A random int in this range is chosen each
episode
    :param fps: FPS limit
    :param save_best: Whether to save the best game
    :param old_actions: True = relative movement, False = absolute movement
    :param mask: Whether to mask instantly-losing actions
    :return: Mean score of the evaluation episodes
    """
    # Extract network from an agent if given
    if isinstance(model, DQNAgent):
        model = model.model
    # Construct the relevant class for the action space
    if old_actions:
        env = Snake3((model.input_shape[1], model.input_shape[2]),
            loop_threshold if not isinstance(model, HumanPlayer) else -1)
    else:
        env = Snake((model.input_shape[1], model.input_shape[2]),
            loop_threshold if not isinstance(model, HumanPlayer) else -1)
    env.fps = fps # Set fps
    # Initialize score tracking infrastructure
    scores = []
    max_score = 0

```



```

# Main game loops
for episode in range(1, episodes + 1):
    # Reset env and obtain the initial state
    current_state = env.reset(random.randint(start_lengths[0], start_lengths[1]),
                               random.randint(foods[0], foods[1]))

    done = False
    if env.is_render: sleep(1) # Wait before starting gameplay
    if save_best: game_log = [current_state] # Initialize game recording
    # Game loop
    while not done:
        # Determine action
        qs = model.predict(tensor_reshape(current_state), verbose=0)
        if mask: qs = mask_loss(current_state, env.body, qs)
        action = np.argmax(qs, axis=-1)
        # Clean user input
        if env.is_render: pygame.event.get()
        new_state, reward, done = env.step(action) # Perform action
        current_state = new_state # Update state
        if save_best: game_log.append(current_state) # Append game recording
    scores.append(env.score) # Save score
    print(f'Game {episode}, Score: {scores[-1]}')
    if save_best:
        game_log.insert(0, env.score) # Add score to the record as metadata
        # Save recording if it's a new high score and wasn't done by a human
        if not isinstance(model, HumanPlayer) and game_log[0] > max_score:
            max_score = game_log[0]
            if not os.path.exists(f'{env.HEIGHT}x{env.WIDTH} best game.pkl') or \
                logs.read_log(f'{env.HEIGHT}x{env.WIDTH} best game')[0] < max_score:
                logs.log_data(game_log, f'{env.HEIGHT}x{env.WIDTH} best game')
                print('saved!')
    if env.is_render: sleep(1) # Wait on final state

return np.mean(np.array(scores))

```

score_evaluation – מבחן ללא אנימציה בסביבה וקטורית:

```
def score_evaluation(model, episodes: int = 100, loop_threshold: int = 200, foods: tuple[int, int] = (1, 1),
                    start_lengths: tuple[int, int] = (4, 4), save_best: bool = False,
                    graph_save_name: str | None = None, old_actions: bool = False) -> float:
    """
    :param model: Either a Q-network or an agent
    :param episodes: All episodes will run in parallel
    :param loop_threshold: Negative value for no loop detection
    :param foods: (min foods, max foods). A random int in this range will be chosen each episode
    :param start_lengths: (min start length, max start length). A random int in this range is chosen each
    episode
    :param save_best: Whether to save a recording of the best game
    :param graph_save_name: Score distribution graph won't be saved if None
    :param old_actions: True = relative movement, False = absolute movement
    :return: Mean score of the evaluation episodes
    """
    # Extract network from an agent if given
    if isinstance(model, DQNAgent):
        model = model.model
    # Construct the relevant class for the action space
    if old_actions:
        vec_env = VecSnake3((model.input_shape[1], model.input_shape[2], episodes), loop_threshold, False,
False)
    else:
        vec_env = VecSnake((model.input_shape[1], model.input_shape[2], episodes), loop_threshold, False,
False)
    # Reset env and obtain the initial states
    current_states = vec_env.reset_all([random.randint(start_lengths[0], start_lengths[1]) for i in
range(episodes)],
[ random.randint(foods[0], foods[1]) for i in range(episodes)])

    dones = [None] # Initialize with some value to enter the main loop
    # Initialize recording if needed
    if save_best:
        game_logs = [[state] for state in current_states]
        best_log = [0]
    # Main loop of all parallel games
    while dones:
        # Determine and perform actions
        actions = np.argmax(model.predict(tensor_reshape(current_states, is_batch=True), verbose=0), axis=-1)
        new_states, rewards, dones = vec_env.step(actions)
        # Remove done environments
        for i in range(len(dones) - 1, -1, -1):
            if dones[i]:
                # Keep recording if it's a new high score, else discard it
                if save_best:
                    if vec_env.active_envs[i].score > best_log[0]:
                        best_log = [vec_env.active_envs[i].score] + game_logs.pop(i)
                else:
                    game_logs.pop(i)
                dones.pop(i)
                new_states.pop(i)
                vec_env.close_one(i)
```

```

        vec_env.active_envs.pop(i)

        current_states = np.array(new_states) # Update states
        if save_best: # Append recordings
            for i in range(len(game_logs)): game_logs[i].append(current_states[i])
        # Save recording of the best game if it's a new high score
        if save_best:
            if not os.path.exists(f'{model.input_shape[1]}x{model.input_shape[2]} best game.pkl') or \
                logs.read_log(f'{model.input_shape[1]}x{model.input_shape[2]} best game')[0] < best_log[0]:
                logs.log_data(best_log, f'{model.input_shape[1]}x{model.input_shape[2]} best game')
                print(f'saved a game with a score of {best_log[0]}!')
        # Save score distribution
        if graph_save_name is not None:
            scores_dist = np.zeros(max(vec_env.scores()) + 1).astype(int)
            for score in vec_env.scores():
                scores_dist[score] += 1
            # Smoothen data
            moving_average = 10
            smooth_scores_dist = np.convolve(scores_dist, np.ones(moving_average) / moving_average, mode='valid')
            # Define graph configuration according to the format
            graph_data = [['score', 'number of games',
                           f'score distribution of {episodes} evaluation episodes
                           ({model.input_shape[1]}x{model.input_shape[2]})'],
                           [[x for x in range(len(scores_dist))], scores_dist, 'Raw', 0.2],
                           [[x for x in range(moving_average // 2, len(smooth_scores_dist) + moving_average // 2)],
                            smooth_scores_dist, f'Moving average ({moving_average})']]
            # Generate and save graph
            graphs.new_graph(graph_data, False, graph_save_name)

    return np.mean(np.array(vec_env.scores()))

```

logs.py

מודול לשמירת נתוני האימון, עוטף את המודול pickle לשם הנוחות.

כולל את הפונקציות `read_log`, `log_data`.

ייבוא ספריות לקובץ logs:

```
import pickle
```

read_log – קריאה מקובץ:

```
def read_log(log_name: str):  
    with open(log_name + '.pkl', 'rb') as log_file:  
        data = pickle.load(log_file)  
    return data
```

log_data – שמירה בקובץ:

```
def log_data(data, log_name: str) -> None:  
    with open(log_name + '.pkl', 'wb') as log_file:  
        pickle.dump(data, log_file)
```

graphs.py

מודול לייצור ושמירת גרפים לנתוני האימון, משתמש ב-matplotlib.

כולל את הפונקציות `new_graph`, `smoothen`, `plot_training_metrics`.

ייבוא ספריות לקובץ `graphs`:

```
import numpy as np
from matplotlib import pyplot as plt

# Installing matplotlib on interpreter interferes with docstring display in the IDE,
# causing :param & :return to be shown as plain text as well as formatted text.
# Deleting matplotlib doesn't help, the only solution is to not install it at first place.
# Matplotlib is the most comfortable graphs package I currently know,
# but a decent alternative might be worth switching to.
```

`new_graph` – ייצור ושמירת גרף:

```
def new_graph(data, view: bool, save_name: str = None) -> None:
    """
    Required data format:

    data = [general labels, line, ..., line]

    general labels = [x-axis label, y-axis label, title]
    line = [[xs], [ys], label, opacity (optional)]

    :param data: According to the specified format
    :param view: whether to display the graph on creation
    :param save_name: None = won't save graph. other = will be saved as png with the given name
    """
    fig, ax = plt.subplots() # Create a new plot
    # Set texts
    labels = data[0]
    ax.set_xlabel(labels[0])
    ax.set_ylabel(labels[1])
    ax.set_title(labels[2])
    # Add all the lines
    for line in data[1:]:
        if len(line) == 4:
            xs, ys, label, opacity = line
        else:
            opacity = 1
            xs, ys, label = line
        ax.plot(xs, ys, label=label, alpha=opacity)

    ax.legend() # Add legend
    if save_name is not None: plt.savefig(save_name + '.png') # Save if needed
    if view: plt.show() # Display if needed
    plt.close()
```

smoothen – החלקת הנתונים באמצעות ממוצע נע:

```
def smoothen(data, filter_size):
    """
    :param data: Single dimensional sequence
    :param filter_size: Size of Moving average filter
    :return:
    """
    return np.convolve(data, np.ones(filter_size) / filter_size, mode='valid')
```

plot_training_metrics – ייצור ושמירת גרפים למדדי אימון:

```
def plot_training_metrics(metrics: list, filter_size: int, save_prefix: str | None = None):
    """
    :param metrics: [[name, data], ...]
    :param filter_size: Size of Moving average filter
    :param save_prefix: Graphs will be saved as {prefix} + {metric} + .png
    :return:
    """
    for metric, data in metrics:
        episodes = [game for game in range(len(data))] # Generate x values
        # Define data according to the format
        plot_data = [['episodes', metric, f'{metric} over training episodes'],
                     [episodes, data, 'Raw', 0.2],
                     [episodes[filter_size - 1:], smoothen(data, filter_size),
                      f'Moving average ({filter_size})']]
        # Generate graph and save if needed
        if save_prefix is None:
            new_graph(plot_data, True)
        else:
            new_graph(plot_data, False, save_prefix + ' ' + metric)
```

emails.py

מודול לשליחת הודעות אימייל, משמש לשליחת עדכונים בזמן אימון.

מכיל רק את הפונקציה `send_message`.

ייבוא ספריות לקובץ emails:

```
from mytime import now # own module
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders
import os
```

`send_message` – שליחת הודעת אימייל (השתמשי במימושים של מקורות [7], [8]):

```
def send_message(subject: str, body: str, filenames: list[str] = None):
    # Define sender and recipient
    receiver_email = "example@gmail.com"
    sender_email = "example@gmail.com"
    sender_password = "example" # Use an App Password if 2FA is enabled
    # Define email message object
    msg = MIMEMultipart()
    msg['From'] = sender_email
    msg['To'] = receiver_email
    msg['Subject'] = subject
    # Attach body
    msg.attach(MIMEText(f'{now()}\n\n' + body, 'plain'))
    # Attach files if given
    if filenames is not None:
        for filename in filenames:
            with open(filename, 'rb') as attachment:
                part = MIMEBase('application', 'octet-stream')
                part.set_payload(attachment.read())
                encoders.encode_base64(part)
                part.add_header(
                    'Content-Disposition',
                    f'attachment; filename= {os.path.basename(filename)}',
                )
            msg.attach(part)
    # Send email
    try:
        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.starttls()
        server.login(sender_email, sender_password)
        server.sendmail(sender_email, receiver_email, msg.as_string())
        server.quit()
        print("Message sent successfully!")
    except Exception as e:
        print(f"Failed to send message: {e}")
```

mytime.py

מודול זמן העוטף את המודול datetime. משמש להערכת זמן לסיום אימונים, לחישוב זמן ריצה, ולפלט והודעות בזמן אימון.

מכיל את הפונקציות `now`, `add_time`, `runtime`.

ייבוא ספריות לקובץ mytime:

```
from datetime import datetime, timedelta
```

`now` – תאריך ושעה נוכחיים:

```
def now() -> datetime:
    """
    :return: Current datetime rounded-up to the next full second
    """
    time_now = datetime.now()
    return time_now + timedelta(microseconds=1_000_000 - time_now.microsecond)
```

`add_time` – חישוב חותמת זמן עתידית:

```
def add_time(delta: timedelta) -> datetime:
    """
    :param delta: A time period
    :return: A future datetime, which is the sum of now and the given time delta
    """
    return now() + delta
```

`runtime` – חישוב זמן ריצה מחותמת זמן:

```
def runtime(start: datetime) -> timedelta:
    """
    :param start: A past timestamp
    :return: The duration since the given timestamp
    """
    return now() - start
```


training.py

קובץ האימון הראשי המשתמש בכל המודולים שיצרתי.

מכיל את הפונקציות *exploration*, *update_epsilon*, *interact*, *train_agent*, *main*.

ייבוא ספריות לקובץ training:

```
# Own modules
import logs
import emails
import graphs
from environments import VecSnake, VecSnake3
from agents import DQNAgent, tensor_reshape
from evaluation import score_evaluation
from mytime import now, add_time, runtime

import random
import numpy as np
from numpy.typing import NDArray
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2' # Reduce tensorflow's verbose
import tensorflow as tf
import multiprocessing as mp
# For profiling
import cProfile
import pstats
import io
```

הגדרת קונפיגורציה לפרמטרים ופיצ'רים (פירוט בייעול אימונים ובהיפר-פרמטרים וכוונן):

```
profile: bool = True
multi_processing: bool = False # Causes retracing in TensorFlow when pygame is present
save_log: bool = True # Evaluation mean scores
save_metrics: bool = True # Training scores, episode lengths, and total rewards
save_graph: bool = True # Happens anyway if summary email is active
verbose: bool = True
summary_email: bool = True
live_emails: bool = True

MODEL_NAME = '15x15 4'
OLD_MODEL_NAME = ''
# Parameters of only one model as an example for מדריך למפתח, the code contains more
parameters = \
{
    '15x15 4': {'OLD_ACTIONS': False,
                'HEIGHT': 15,
                'WIDTH': 15,
                'MIN_FOODS': 1,
                'MAX_FOODS': 1,
                'MIN_START_LENGTH': 4,
```

```

        'MAX_START_LENGTH': 4,
        'ROUNDS': 100,
        'FITS': 1000,
        'ENVS': 100,
        'FIT_FREQ': 1,
        'EPOCHS_PER_FIT': 1,
        'MINIBATCH_SIZE': 1000,
        'MEMORY_SIZE': 1_000_000,
        'MIN_MEMORY_SIZE': 100_000,
        'TARGET_SYNC_FREQ': 100,
        'TAU': 1,
        'GAMMA': 0.95,
        'EPSILON': 1,
        'EPSILON_DECAY': 0.999925,
        'MIN_EPSILON': 0.01,
        'DECAY_BEFORE_TRAINING': True},
    }

```

טעינת פרמטרים (פירוט בהיפר-פרמטרים וכוונות):

```

# Loading of training parameters
OLD_ACTIONS: bool = parameters[MODEL_NAME]['OLD_ACTIONS']
HEIGHT: int = parameters[MODEL_NAME]['HEIGHT']
WIDTH: int = parameters[MODEL_NAME]['WIDTH']
MIN_FOODS: int = parameters[MODEL_NAME]['MIN_FOODS']
MAX_FOODS: int = parameters[MODEL_NAME]['MAX_FOODS']
MIN_START_LENGTH: int = parameters[MODEL_NAME]['MIN_START_LENGTH']
MAX_START_LENGTH: int = parameters[MODEL_NAME]['MAX_START_LENGTH']
ROUNDS: int = parameters[MODEL_NAME]['ROUNDS']
FITS: int = parameters[MODEL_NAME]['FITS'] # For every .fit, {ENVS * FIT_FREQ} steps are done
ENVS: int = parameters[MODEL_NAME]['ENVS'] # Vectorized. Amount of parallel envs
FIT_FREQ: int = parameters[MODEL_NAME]['FIT_FREQ']
EPOCHS_PER_FIT: int = parameters[MODEL_NAME]['EPOCHS_PER_FIT'] # 1 is probably always the best choice
MINIBATCH_SIZE: int = parameters[MODEL_NAME]['MINIBATCH_SIZE']
MEMORY_SIZE: int = parameters[MODEL_NAME]['MEMORY_SIZE']
MIN_MEMORY_SIZE: int = parameters[MODEL_NAME]['MIN_MEMORY_SIZE']
TARGET_SYNC_FREQ: int = parameters[MODEL_NAME]['TARGET_SYNC_FREQ'] # Amount of .fits between updates
TAU: float = parameters[MODEL_NAME]['TAU'] # 1 = hard update
GAMMA: float = parameters[MODEL_NAME]['GAMMA']
epsilon: float = parameters[MODEL_NAME]['EPSILON']
EPSILON_DECAY: float = parameters[MODEL_NAME]['EPSILON_DECAY']
MIN_EPSILON: float = parameters[MODEL_NAME]['MIN_EPSILON']
DECAY_BEFORE_TRAINING: bool = parameters[MODEL_NAME]['DECAY_BEFORE_TRAINING']

# File management
SAVE_NAME = MODEL_NAME + '.keras'
LOG_NAME = MODEL_NAME + ' log'

# Define metrics globally for usage in interact()
if os.path.exists(MODEL_NAME + ' metrics.pkl'):
    metrics = logs.read_log(MODEL_NAME + ' metrics')
else:
    metrics = [['score', []], ['total_reward', []], ['episode_length', []]]

```

exploration – האם לחקור פעולה חדשה:

```
def exploration() -> bool:
    """
    Applies epsilon-greedy policy
    :return: Whether to explore
    """
    return random.random() < epsilon
```

update_epsilon – הפחתת אפסילון:

```
def update_epsilon() -> None:
    """
    Applies one exponential decay update to epsilon
    """
    global epsilon
    if epsilon > MIN_EPSILON:
        epsilon *= EPSILON_DECAY
```

interact – אינטראקציה עם הסביבה (ייצור נתונים):

```
def interact(interaction_model, vec_env: VecSnake, steps: int) -> list[
    tuple[NDArray[float], int, int, NDArray[float], bool]]:
    """
    :param interaction_model: A keras neural network
    :param vec_env: A vectorized environment with attribute continuous=True
    :param steps: Amount of interaction steps
    :return: A list of experiences
    """
    experiences = [] # Initialize interaction storage data structure
    current_states = vec_env.get_states() # Retrieve initial states
    # Interaction loop
    for step in range(steps):
        qs = interaction_model.predict(tensor_reshape(current_states, is_batch=True), verbose=0) # Predict Q values
        actions = np.argmax(qs, axis=-1) # Extract the actions with maximal Q values
        actions = [random.randint(0, 3) if exploration() else action for action in actions] # Explore
        new_states, rewards, dones = vec_env.step(actions) # Perform interaction
        experiences += [experience for experience in
            zip(current_states, actions, rewards, new_states, dones, strict=True)] # Store experiences
        current_states = np.array(new_states) # Update states for the next step
    # Reset done environments
    for i in range(vec_env.size):
        if dones[i]:
            # Store game metrics
            metrics[0][-1].append(vec_env.scores(i))
            metrics[1][-1].append(vec_env.total_rewards(i))
            metrics[2][-1].append(vec_env.episode_lengths(i))
            # Reset
            current_states[i] = vec_env.reset_one(i, random.randint(MIN_START_LENGTH, MAX_START_LENGTH),
                random.randint(MIN_FOODS, MAX_FOODS))
    return experiences
```

train_agent – לולאת אימון ראשית:

```

def train_agent() -> None:
    # ----- Initialization Sequence -----
    # Load a trained model for enlargement
    if OLD_MODEL_NAME != '':
        old_model = tf.keras.models.load_model(OLD_MODEL_NAME + '.keras')
    else:
        old_model = None
    # Initialize agent
    agent = DQNAgent(SAVE_NAME, MEMORY_SIZE, (HEIGHT, WIDTH), old_model)
    interaction_model = tf.keras.models.clone_model(agent.model)
    interaction_model.set_weights(agent.model.get_weights())
    # Initialize environment
    if OLD_ACTIONS:
        vec_env = VecSnake3((agent.model.input_shape[1], agent.model.input_shape[2], ENVS), render=False)
    else:
        vec_env = VecSnake((agent.model.input_shape[1], agent.model.input_shape[2], ENVS), render=False)
    vec_env.reset_all([random.randint(MIN_START_LENGTH, MAX_START_LENGTH) for i in range(ENVS)],
                      [random.randint(MIN_FOODS, MAX_FOODS) for i in range(ENVS)])
    # Define evaluation eval_data structure according to format. Used for human monitoring and logging.
    if os.path.exists(LOG_NAME + '.pkl'): # Continue an existing log
        eval_data = logs.read_log(LOG_NAME)
        past_iterations = eval_data[1][0][-1]
        eval_data[0][-1] = f'{MODEL_NAME} evaluation score over {past_iterations + FITS * ROUNDS} training iterations'
    else: # Create a new log
        past_iterations = 0
        if verbose: print(f'{now()} : starts evaluating round 0 / {ROUNDS}')
        eval_data = [[f'training iterations', 'mean score',
                      f'{MODEL_NAME} evaluation score over {FITS * ROUNDS} training iterations'],
                     [[0], [score_evaluation(agent.model)], MODEL_NAME]]
    # Gather initial experience
    if verbose: print(f'{now()} : starts gathering initial experience')
    while len(agent.memory) < MIN_MEMORY_SIZE:
        experiences = interact(interaction_model, vec_env, FIT_FREQ) # Generate experiences
        agent.update_memory(experiences) # Append experience buffer
        # Decay epsilon if needed
        if DECAY_BEFORE_TRAINING:
            update_epsilon()
    if verbose:
        print(f'{now()} : finished gathering initial experience')
        print(f'{now()} : starts {FITS * ROUNDS} training iterations. evaluation every {FITS}')
    # Initialize multi-processing if needed
    if multi_processing:
        pool = mp.Pool(processes=1)

    # ----- Main Training Loop -----
    for r in range(1, ROUNDS + 1):
        round_start = now() # Save start time for runtime estimation calculation
        if verbose: print(f'{now()} : starts training round {r} / {ROUNDS}')
        # Training iterations
        for n in range(1, FITS + 1):
            if multi_processing:
                interaction = pool.starmap_async(interact, [
                    (interaction_model, vec_env, FIT_FREQ)]) # Start asynchronous interaction (data generation)
                agent.fit_minibatch(MINIBATCH_SIZE, EPOCHS_PER_FIT, GAMMA, TAU,
                                    TARGET_SYNC_FREQ) # Train on previously collected experiences

```

```

        experiences = interaction.get()[0] # Wait for interaction to finish
        agent.update_memory(experiences) # Append experience buffer
    else:
        experiences = interact(interaction_model, vec_env, FIT_FREQ) # Generate experiences
        agent.update_memory(experiences) # Append experience buffer
        agent.fit_minibatch(MINIBATCH_SIZE, EPOCHS_PER_FIT, GAMMA, TAU, TARGET_SYNC_FREQ) # Train
        interaction_model.set_weights(agent.model.get_weights()) # Sync models (important for multi-processing)
        update_epsilon() # Decay epsilon to manage exploration/exploitation
# Evaluate and store result
if verbose: print(f'{now()} : starts evaluating round {r} / {ROUNDS}')
eval_data[1][0] += [r * FITS + past_iterations]
eval_data[1][1] += [score_evaluation(agent, old_actions=OLD_ACTIONS)]
# Checkpoint
agent.save(f'round {r} ' + SAVE_NAME) # Save model
if save_log: logs.log_data(eval_data, LOG_NAME) # Save evaluation data
if save_metrics: logs.log_data(metrics, MODEL_NAME + ' metrics') # Save training metrics
estimation = add_time((ROUNDS - r) * runtime(round_start)) # Estimate run finish time
if verbose: print(f'{now()} : evaluation result: {eval_data[1][1][-1]} estimated finish: {estimation}')
if live_emails: # Send a training status email
    graphs.new_graph(eval_data, False, LOG_NAME)
    emails.send_message(f'round {r}/{ROUNDS} live update - {LOG_NAME}',
                        f'Estimated finish: {estimation}\nRaw evaluation eval_data so far:\n{eval_data}',
                        [LOG_NAME + '.png'])

# ----- Finalization Sequence -----
agent.save(SAVE_NAME) # Save model
if save_log: logs.log_data(eval_data, LOG_NAME) # Save evaluation data
if save_metrics: logs.log_data(metrics, MODEL_NAME + ' metrics') # Save training metrics
if summary_email:
    # Define email content
    subject = MODEL_NAME + ' training summary'
    body = (f'Finished training {MODEL_NAME}\n'
            f'Training started at {start} and finished at {now()} with a total runtime of {runtime(start)}\n'
            f'This email contains the following attachments:\n'
            f'- the trained model\n'
            f'- evaluation & training metrics\n'
            f'- graphs of the metrics')
    attachments = [SAVE_NAME, LOG_NAME + '.pkl', f'{MODEL_NAME} metrics.pkl', f'{MODEL_NAME} log.png',
                   f'{MODEL_NAME} score.png', f'{MODEL_NAME} episode_length.png', f'{MODEL_NAME}
total_reward.png']
    # Smoothen evaluation data
    eval_data[1][2] = 'Raw'
    eval_data[1].append(0.3)
    moving_average_size = 5
    eval_data.append(
        [eval_data[1][0][moving_average_size - 1:], graphs.smoothen(eval_data[1][1], moving_average_size),
         f'Moving average ({moving_average_size})'])
# Generate & save graphs
graphs.new_graph(eval_data, False, LOG_NAME)
graphs.plot_metrics(metrics, 1000, MODEL_NAME)
# Send email
emails.send_message(subject, body, attachments)
elif save_graph:
    # Smoothen evaluation data
    eval_data[1][2] = 'Raw'
    eval_data[1].append(0.3)
    moving_average_size = 5

```

```

eval_data.append(
    [eval_data[1][0][moving_average_size - 1:], graphs.smoothen(eval_data[1][1], moving_average_size),
     f'Moving average ({moving_average_size})'])
# Generate & save graphs
graphs.new_graph(eval_data, False, LOG_NAME)
graphs.plot_metrics(metrics, 1000, MODEL_NAME)
if verbose: print(f'{now()} : done training')

```

main ותחילת הריצה:

```

def main():
    train_agent()

if __name__ == '__main__':
    start = now() # Save start time for runtime calculation
    if profile:
        # Profile using python's official documentation's code example
        profiler = cProfile.Profile()
        profiler.enable()
        main()
        profiler.disable()
        s = io.StringIO()
        ps = pstats.Stats(profiler, stream=s).sort_stats(pstats.SortKey.TIME)
        ps.print_stats()
        # Save
        with open(MODEL_NAME + ' profile', 'w') as save_file:
            save_file.write(s.getvalue())
            save_file.close()
        print(MODEL_NAME + ' profile saved')
    else:
        main()

    print(f'runtime was {runtime(start)}') # Calculate and output total runtime

```

testing.py

קובץ הכולל תוכנית ליישום ומבחן מודלים, ומספק שליטה נוחה על הגדרות רבות.

ייבוא ספריות לקובץ testing:

```

from mytime import now, runtime
import os

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
from tensorflow.keras import models
from evaluation import score_evaluation, visual_evaluation
from agents import HumanPlayer, RandomPlayer

```

שליטה בהגדרות היישום/מבחן:

```

# Deployment settings - only modify this segment
# -----
model_name = '15x15 4' # 'random' for random player, 'human' for human player
render = True # If render = False then all episodes will be played in parallel
fps = 30 # FPS limit; may run slower for extremely large models
episodes = 1000 # Amount of games. -1 for endless evaluation, requires render=True
loop_threshold = -1 # Negative value for no loop detection
foods = (1, 1) # A random int in this range is chosen each episode
start_lengths = (4, 4) # A random int in this range is chosen each episode
save_best = True # Whether to save the best game
old_actions = False # True = relative movement, False = absolute movement
mask = True # Whether to mask instantly-losing actions
height, width = 16, 16 # Only affects human & random player
# -----

```

טעינת המודל:

```

# Load model
match model_name.lower():
    case 'random':
        model = RandomPlayer((1, height, width, 1))
    case 'human':
        model = HumanPlayer((1, height, width, 1))
    case _:
        model = models.load_model(model_name + '.keras')

```

יישום:

```
# Save timestamp for runtime calculation
start_time = now()
# Evaluate
if render:
    mean_score = visual_evaluation(model, episodes, loop_threshold, foods, start_lengths,
                                   fps, save_best, old_actions, mask)
else:
    mean_score = score_evaluation(model, episodes, loop_threshold, foods, start_lengths,
                                   save_best, model_name + ' distribution', old_actions)
# Summarize
print('mean score: ', mean_score)
print(f'runtime was {runtime(start_time)}')
```


view game.py

תוכנית לצפייה במשחקים שנשמרו במהלך מבחן.

ייבוא ספריות לקובץ view game:

```
import os
import time

os.environ['PYGAME_HIDE_SUPPORT_PROMPT'] = '1' # Hide pygame's welcome output
import pygame
import logs
from environments import Snake
```

קביעת הגדרות:

```
# Only modify these 2 lines:
fps = 20
height, width = 15, 15
```

תוכנית לצפייה במשחק:

```
# Load game recording
game = logs.read_log(f'{height}x{width} best game')
print('score:', game[0])
print('shape:', game[1].shape)
# Initialize variables
env = Snake(game[1].shape)
clock = pygame.time.Clock()
flag = True
# Iterate over frames
for frame in game[1:]:
    env.state = frame
    env.render()
    clock.tick(fps)
    pygame.event.get() # Clean user input
    # Wait on first frame before starting
    if flag:
        time.sleep(2)
        flag = False
time.sleep(3) # Wait on final frame
```

מדריך למשתמש

כפי שכבר ציינתי, קהל היעד של הפרויקט הוא בעיקר מפתחים ועבורם קיים מדריך למפתח. עם זאת, גם אנשים שאינם מפתחים יכולים להפעיל את המודלים המאומנים שישחקו את המשחק, ואפילו לשחק בעצמם במימוש שלי למשחק Snake. לאחר הורדת קבצי הפרויקט והספריות המופיעות במדריך למפתח, ניתן לערוך ולהריץ אותם בכל כלי שתמצאו לנכון, כגון PyCharm, Visual Studio Code, Notepad...

הקבצים לשימוש הם `view game.py` ו-`testing.py`.

בקובץ `testing.py` ניתן להפעיל מודלים שמורים ולצפות בהם משחקים, או לבחון אותם ללא אנימציה ולראות את התוצאות. נוסף לכך, ניתן לשחק באמצעות החיצים במקלדת כאשר שם המודל הוא 'human'. להסבר קצר על המשחק, ראה מבוא. ניתן לשנות את ההגדרות בחלק זה של הקובץ בלבד:

```
# Deployment settings - only modify this segment
# -----
model_name = '15x15 4' # 'random' for random player, 'human' for human player
render = True # If render = False then all episodes will be played in parallel
fps = 30 # FPS limit; may run slower for extremely large models
episodes = 1000 # Amount of games. -1 for endless evaluation, requires render=True
loop_threshold = -1 # Negative value for no loop detection
foods = (1, 1) # A random int in this range is chosen each episode
start_lengths = (4, 4) # A random int in this range is chosen each episode
save_best = True # Whether to save the best game
old_actions = False # True = relative movement, False = absolute movement
mask = True # Whether to mask instantly-losing actions
height, width = 16, 16 # Only affects human & random player
# -----
```

בקובץ `view game.py` ניתן לצפות מחדש במשחקים שנשמרו מהמבחן בקובץ `testing.py`. שימו לב: רק המשחק הטוב ביותר נשמר וניתן לצפייה. משחק שיעקוף ניקוד זה בגודל לוח זהה יחליף את המשחק השמור הקודם. ניתן לשלוט בגודל הלוח בהקלטה ובמהירות הניגון (fps), ע"י שינוי השורות הבאות:

```
# Only modify these 2 lines:
fps = 20
height, width = 15, 15
```

רפלקציה

נחשפתי לתחום Reinforcement Learning במקרה בסרטון ביוטיוב ומאוד התחברתי לשיטה, אז התחלתי להתעניין ולקרוא עוד על הנושא. השלמתי את סדרת הקורסים *ML Specialization* ב-Coursera (שכללו חלק על RL). המשכתי להעמיק בלמידה ממקורות שונים באינטרנט, והתמקדתי באלגוריתם DQN שהתחברתי אליו, במיוחד בשל הדמיון הרב בינו לבין למידה של בני אדם ובעלי חיים.

מאוד נהניתי ללמוד את החומר התיאורטי וליישם אותו, למרות כל האתגרים בהם נתקלתי והבאגים שנאבקתי לפתור, בין אם בפיתוח, הבנה ומימוש של אלגוריתמים, ובין אם בהתקנת ספריות, התמודדות עם ביצועים נמוכים, שינויים ותוספות לחלקים שמימשתי בעבר, תכנון המערכת וכו'. זהו פרויקט התוכנה בעל ההיקף הרחב ביותר שאי פעם עשיתי, ועבדתי עליו באופן עצמאי לחלוטין, לכן העבודה כללה היתקלות בקשיים גם מחוץ לעולם למידת המכונה באופן ספציפי – את החשיבות של סדר, תיעוד, חשיבה מערכתית, ומשוב עצמי שוטף נאלצתי להבין בדרך הקשה כשהייתי כבר עמוק בתוך הפרויקט. נוסף לכך, למדתי את החשיבות של סקירת הפתרונות הקיימים לפני שניגשים לפתור בעיה. מצאתי את עצמי משקיע זמן רב בחשיבה על פתרונות ושיטות, ואחר כך מגלה שהפתרון שחשבתי עליו כבר קיים ומקובל. זוהי מהות ההתנסות, אך אם לא מגבילים אותה היא באה על חשבון היקף הפרויקט וקצב ההתקדמות.

התשוקה שלי לתחום הפכה את העבודה על הפרויקט לחוויה מיוחדת במינה, ופעמים רבות מצאתי את עצמי יושב ועובד ימים שלמים בלי הפסקה, ולא מצליח לעצור וללכת לישון גם כשכבר לילה. אין לי ספק שההתנסות בסגנון עבודה זה, והכישורים שפיתחתי לאורך הדרך ישרתו אותי בהמשך חיי. למעשה, הפרויקט תרם לי רבות כבר בשלב זה – מצאתי את עצמי מספר עליו באריכות בראיונות למסלולים בצבא, ואני בטוח שהדבר תרם רבות לכך שבסופו של דבר התקבלתי.

אם הייתי מתחיל לעבוד על הפרויקט היום, הייתי משנה בעיקר את אופן הפיתוח למטרה ארוכת טווח – הייתי קודם כל מנסה להבין לאיזה סדר גודל אני מתכוון להגיע, ומתאים את הקוד שאני כותב להיות עמיד, מובן ומסודר בשביל לאפשר התרחבות ושילוב תוספות באופן הקל והמוצלח ביותר שניתן.

אשמח מאוד לראות המשך עבודה ומחקר בתחום, אפילו המשך של הפרויקט שלי. במיוחד לגבי הגדלת מודלים מאומנים, הייתי רוצה לראות אם באמת ניתן לחזות את ההגדלה הכי יעילה על בסיס אימונים מקדימים או שיטות מתמטיות. כזכור מהניסיונות שלי לעשות זאת באימון מקדים, אין לי את משאבי המחשוב הדרושים לכך. בנוסף, בהמשך הפיתוח הייתי רוצה להוסיף ולבדוק פיצ'רים נוספים של DQN כגון prioritized experience replay, multi-step returns, distributional RL, multi-agent environments.

ביבליוגרפיה

מאמרים:

- [1] Abraham, D. (2023). [Deep reinforcement learning for Snake: How well does a deep Q-network model for classic Snake perform with slight variations in the game rules?](#) [Bachelor's thesis, Eindhoven University of Technology & Tilburg University].
- [2] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2017). [Rainbow: Combining improvements in deep reinforcement learning](#) (arXiv:1710.02298v1). arXiv.
- [3] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). [Playing Atari with deep reinforcement learning](#) (arXiv:1312.5602). arXiv.
- [4] Sewak, M. (2019). [Deep Q Network \(DQN\), Double DQN, and Dueling DQN: A step towards general artificial intelligence](#). In *Deep reinforcement learning* (pp. 95–108). Springer.
- [5] Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2016). [Dueling network architectures for deep reinforcement learning](#) (arXiv:1511.06581v3). arXiv.

מדריכים ואתרים:

- [6] <https://docs.python.org/3/library/profile.html>
- [7] <https://mailtrap.io/blog/smtplib/#Send-email-with-attachments>
- [8] <https://www.geekslovecoding.com/blog/python-email-sending-tutorial/>
- [9] https://en.wikipedia.org/wiki/Q-learning#Deep_Q-learning
- [10] https://www.coursera.org/specializations/machine-learning-introduction?utm_medium=sem&utm_source=gg&utm_campaign=b2c_emea_x_coursera_ftcof_courseraplus_cx_dr_bau_gg_sem_bd-ex_s1_en_m_hyb_24-10_x&campaignid=21836581617&adgroupid=351685084750&device=c&keyword=coursera&matchtype=e&network=g&devicemodel=&creativeid=1449957450621&assetgroupid=&targetid=kwd-36262515261&extensionid=&placement=&gad_source=1&gad_campaignid=21836581617&gbraid=0AAAAADdKX6aPSGzZ81mnjZ5VbFAGCbrBw&gclid=Cj0KCQjww-HABhCGARIsALLO6XwCdml_BBy7ltBaWL1Dhxm1B_BSW-6c5KdbOVbBpHLEXw3NCs6caAsa9EALw_wcB#courses

נספחים

הסבר על RL ו-DQN:

<https://www.baeldung.com/cs/q-learning-vs-deep-q-learning-vs-deep-q-network>

העמקה על שדרוגים של DQN:

Sewak, M. (2019). [Deep Q Network \(DQN\), Double DQN, and Dueling DQN: A step towards general artificial intelligence](#). In *Deep reinforcement learning* (pp. 95–108). Springer.